

DR



DeepRob

Lecture 7
Convolutional Neural Networks
University of Michigan and University of Minnesota



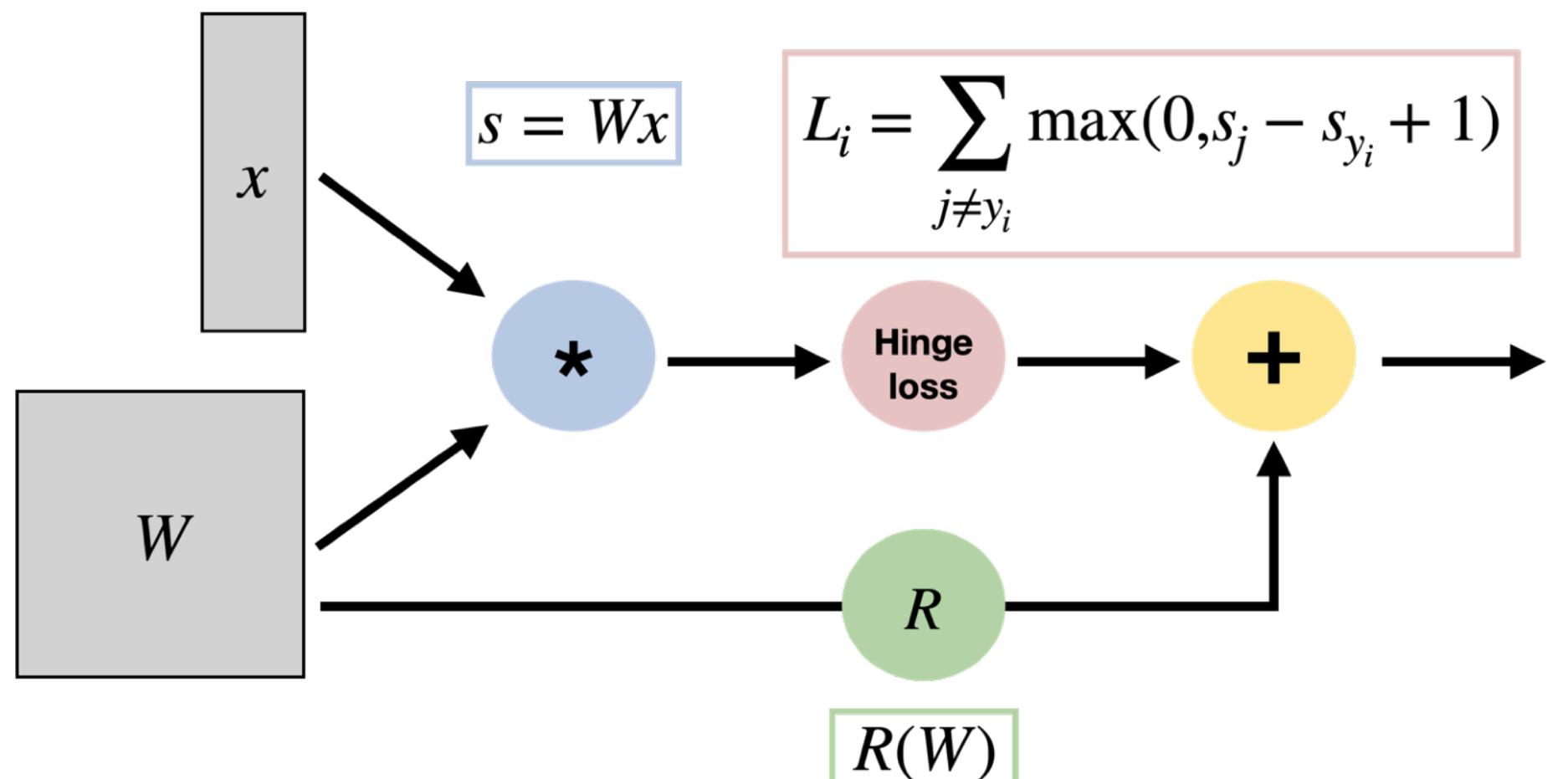
Project 1 – Reminder

- Instructions and code available on the website
 - Here: deeprob.org/projects/project1/
- Uses Python, PyTorch and Google Colab
- Implement KNN, linear SVM, and linear softmax classifiers
- **Autograder is online and updated**
- **Due Thursday, January 26th 11:59 PM EST**



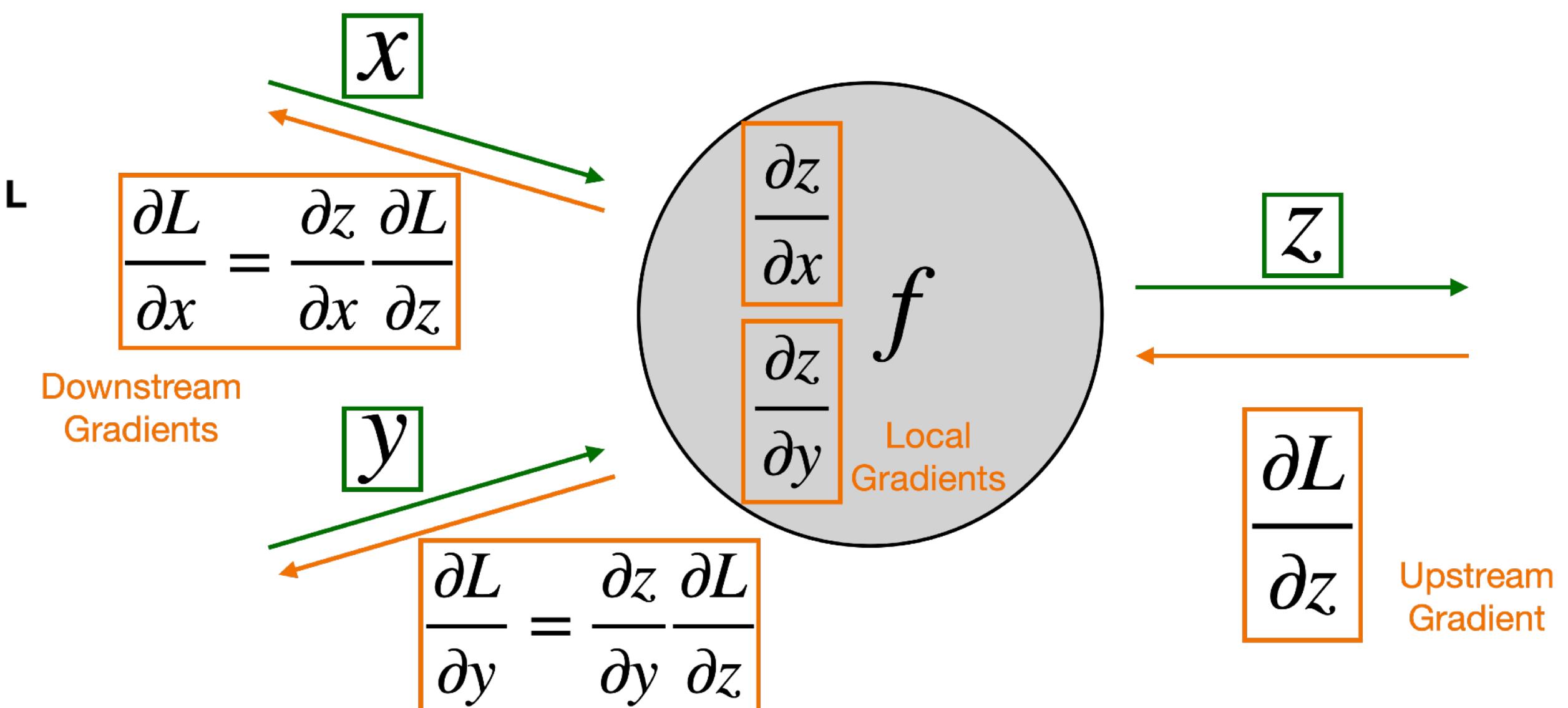
Recap from Previous Lecture

Represent complex expressions as **computational graphs**



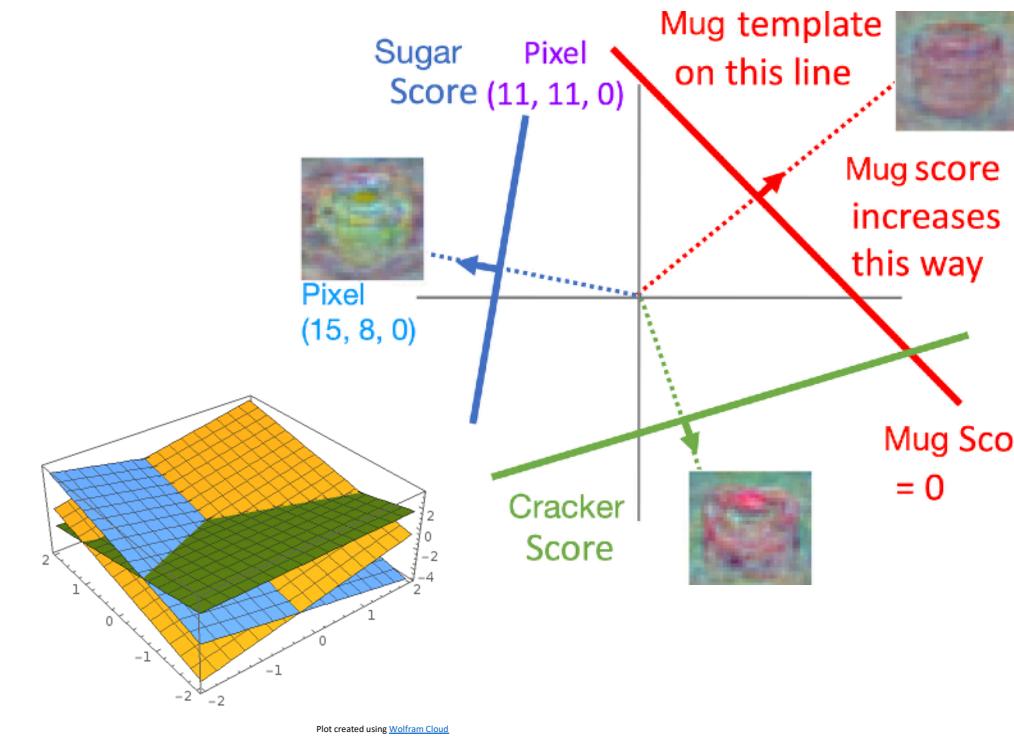
1. Forward pass: Compute outputs

During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients**



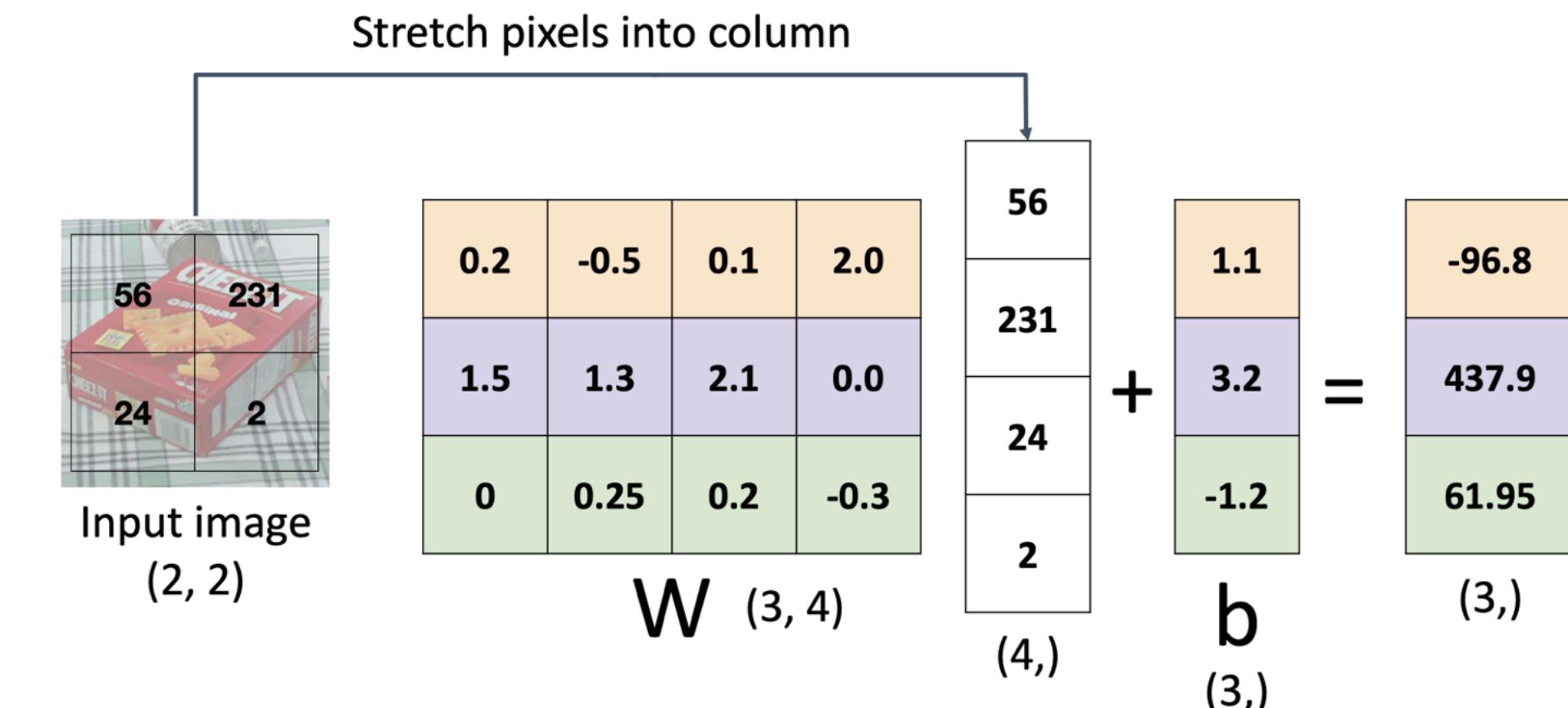
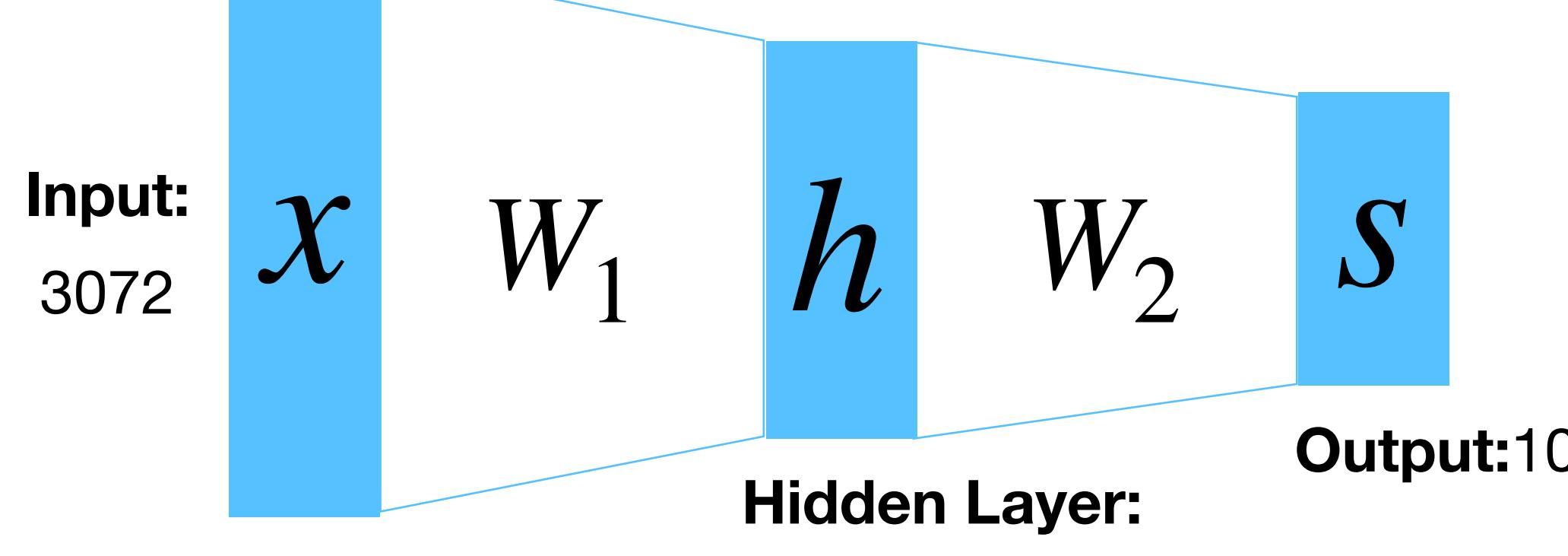
2. Backward pass: Compute gradients

Recap from Previous Lecture

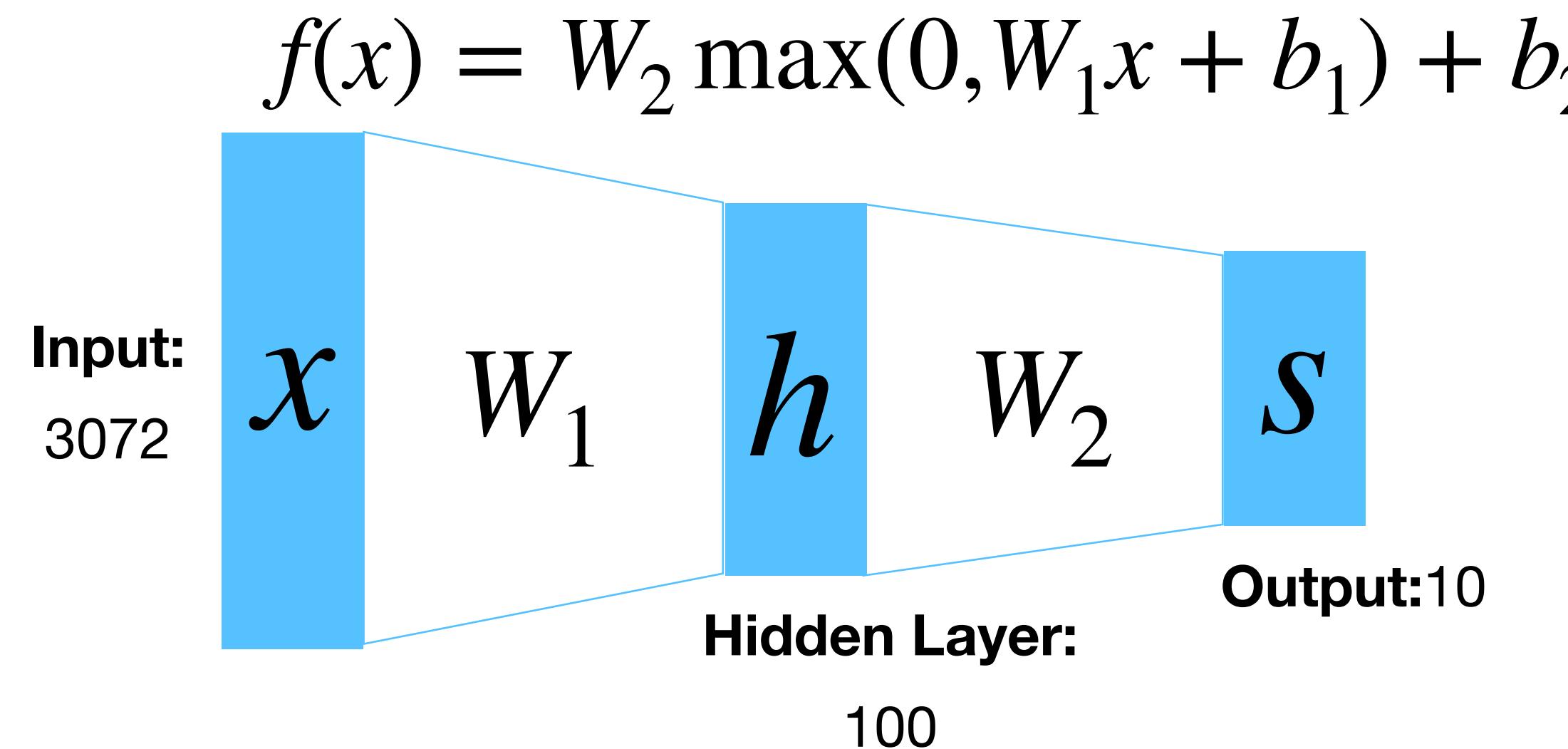
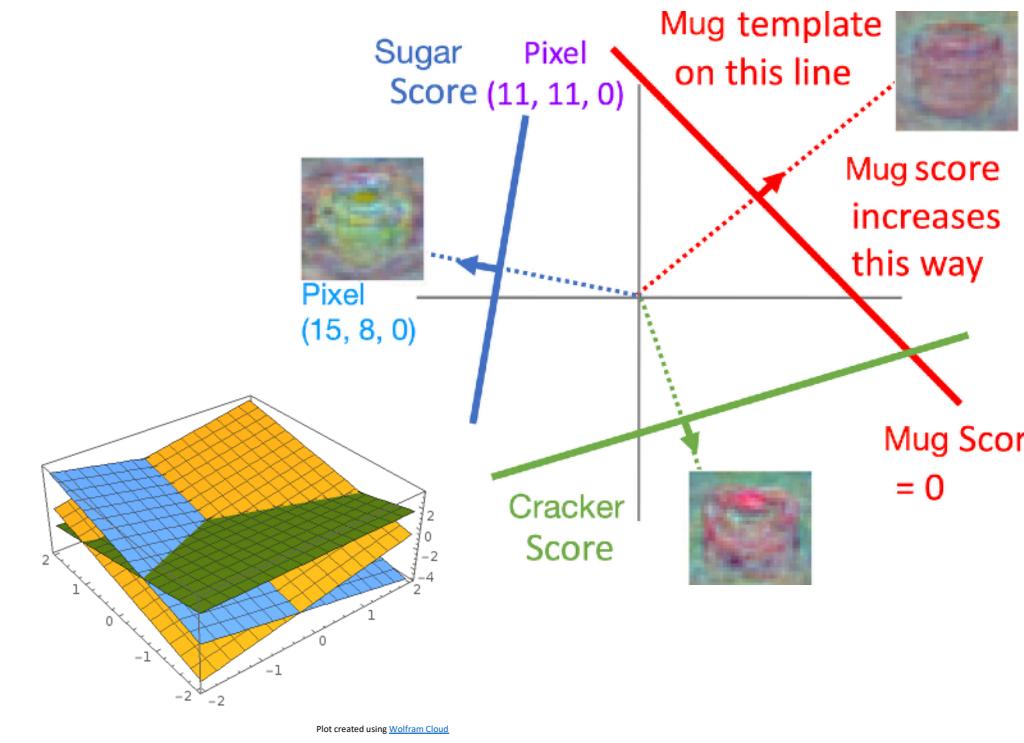


Problem: So far our classifiers don't respect the spatial structure of images!

$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$

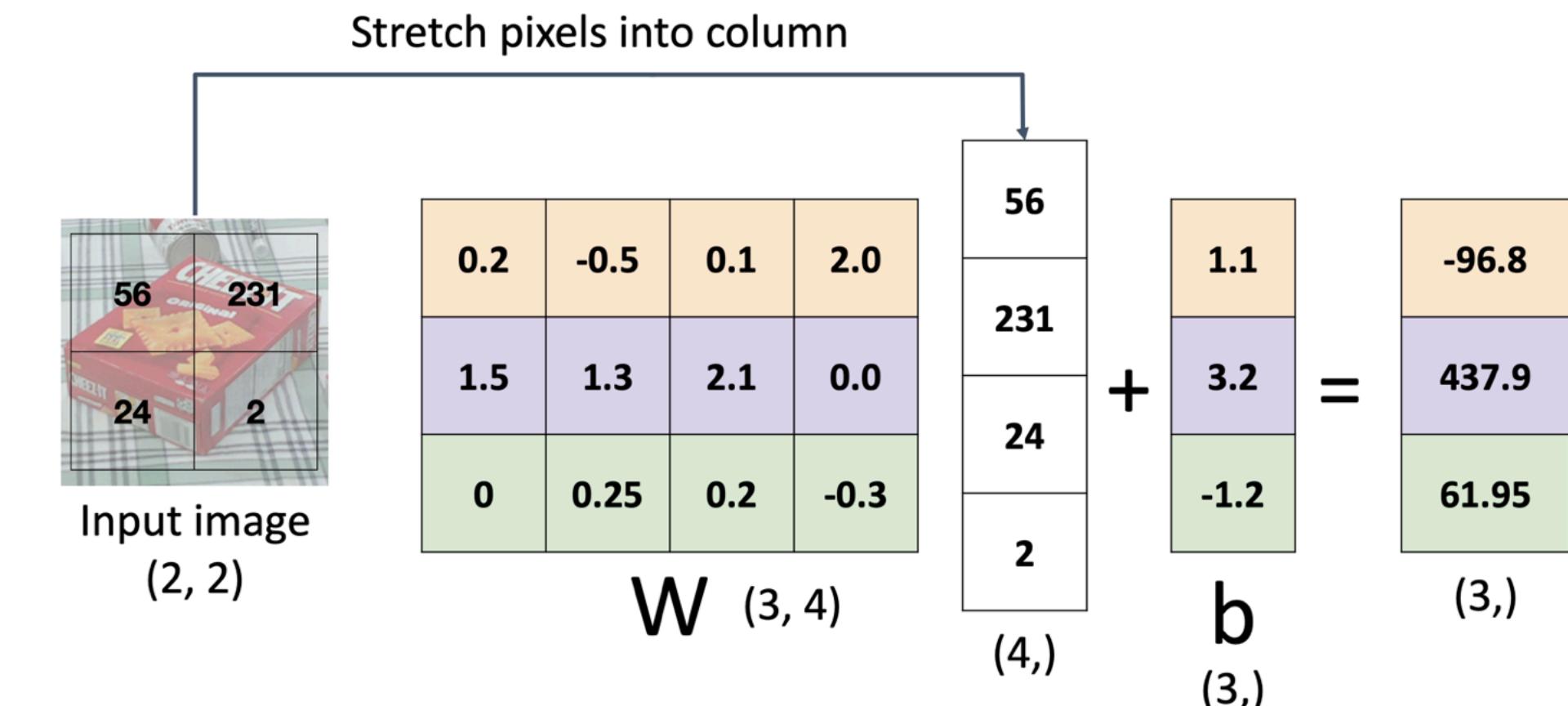


Recap from Previous Lecture



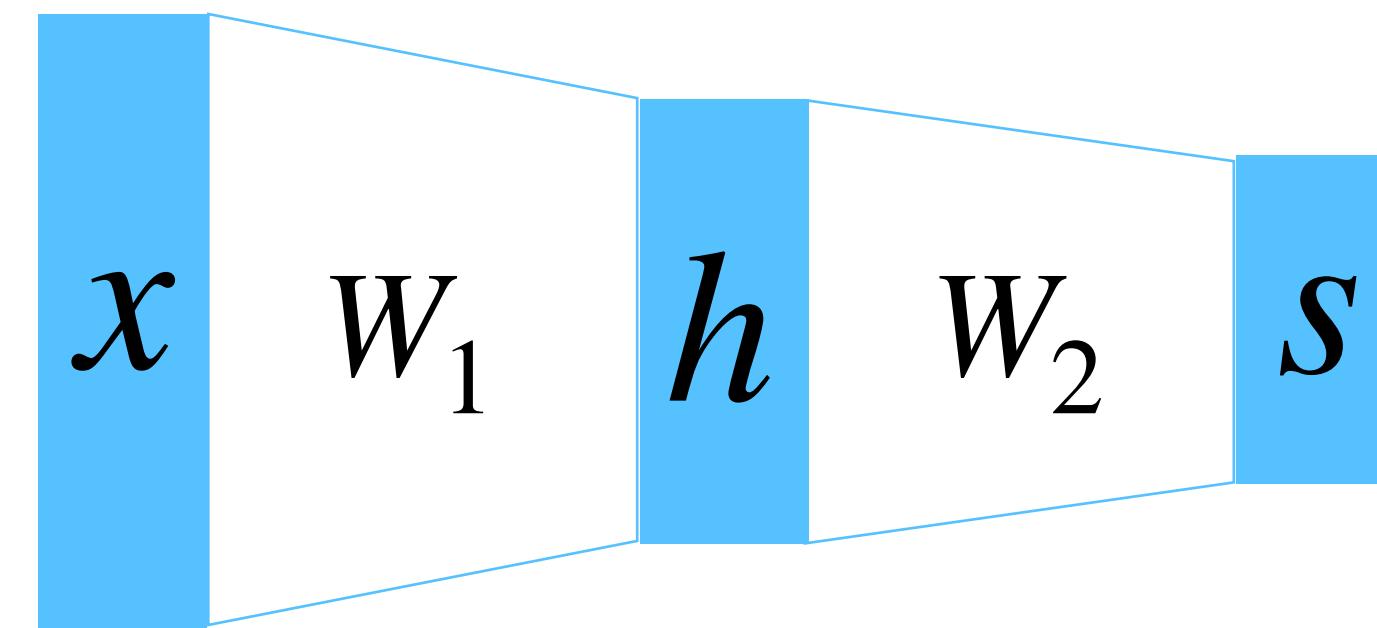
Problem: So far our classifiers don't respect the spatial structure of images!

Solution: Define new computational nodes that operate on images!

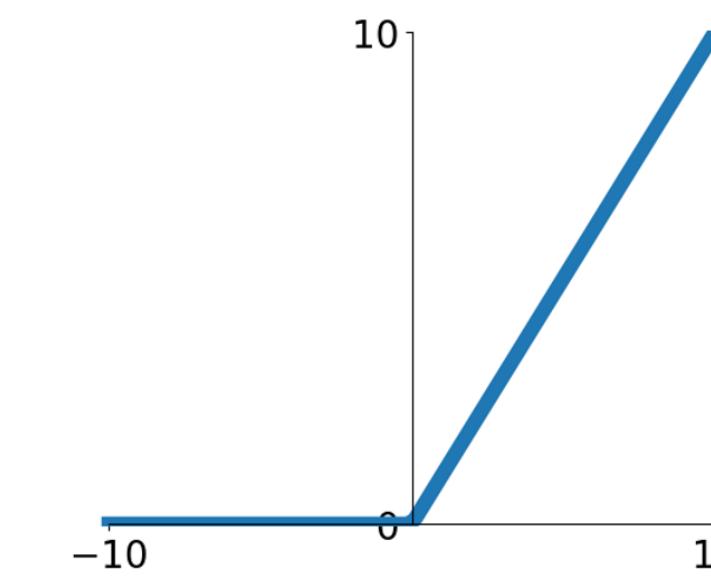


Components of Fully-Connected Networks

Fully-Connected Layers

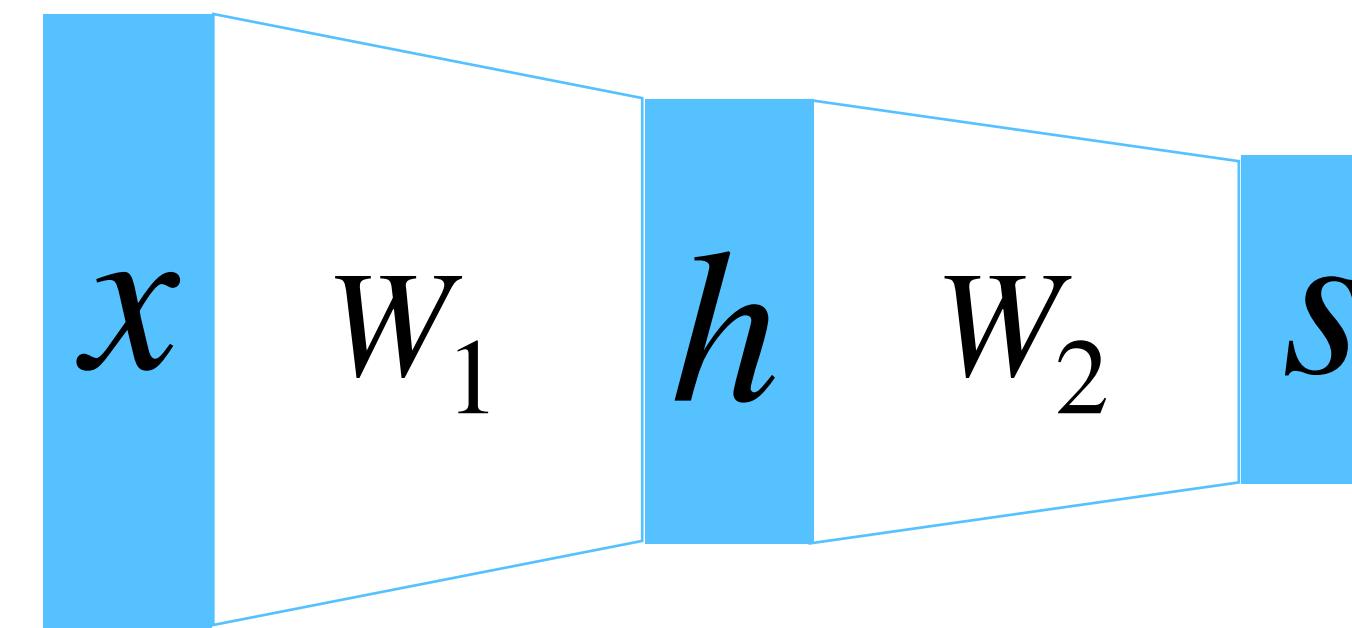


Activation Functions

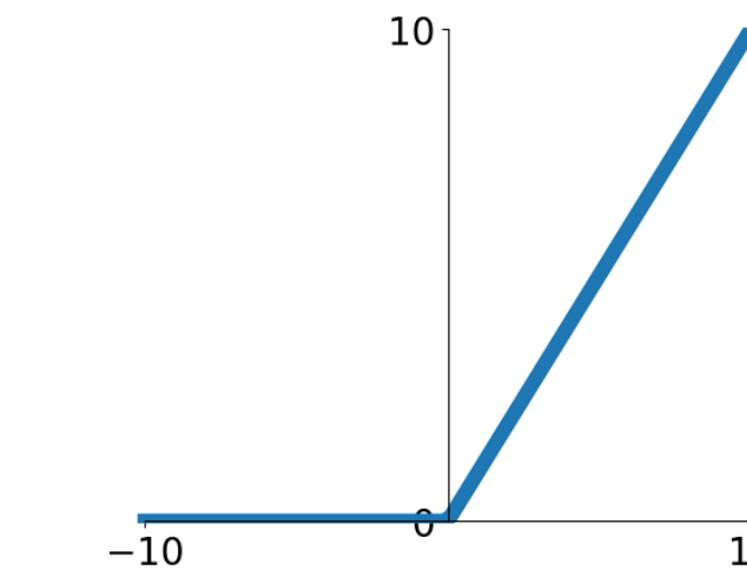


Components of Convolutional Neural Networks

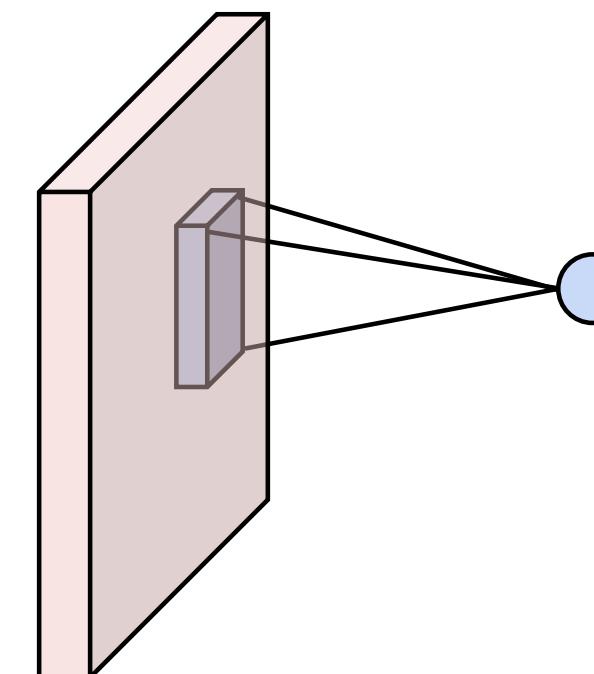
Fully-Connected Layers



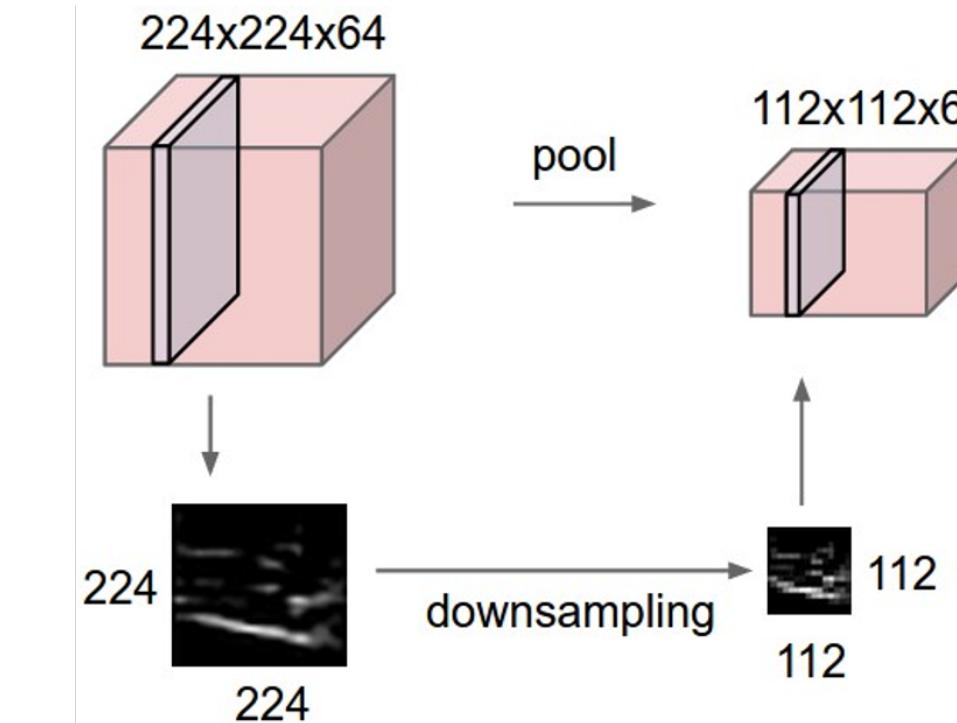
Activation Functions



Convolution Layers



Pooling Layers

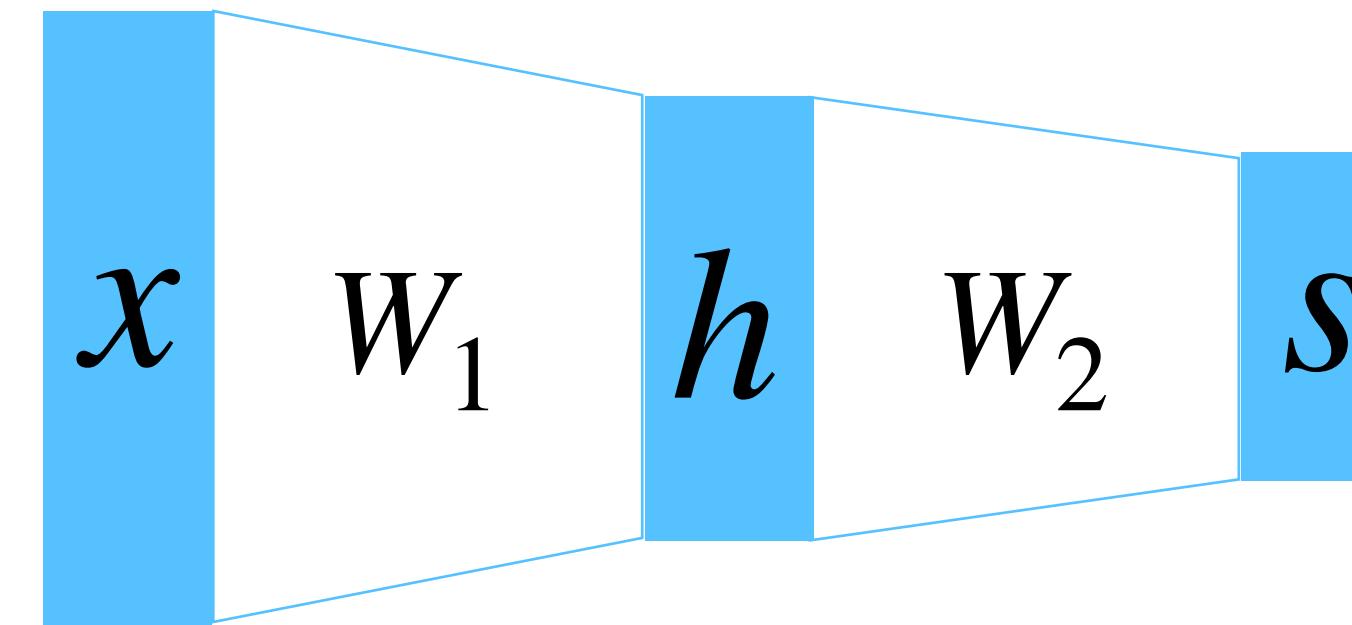


Normalization

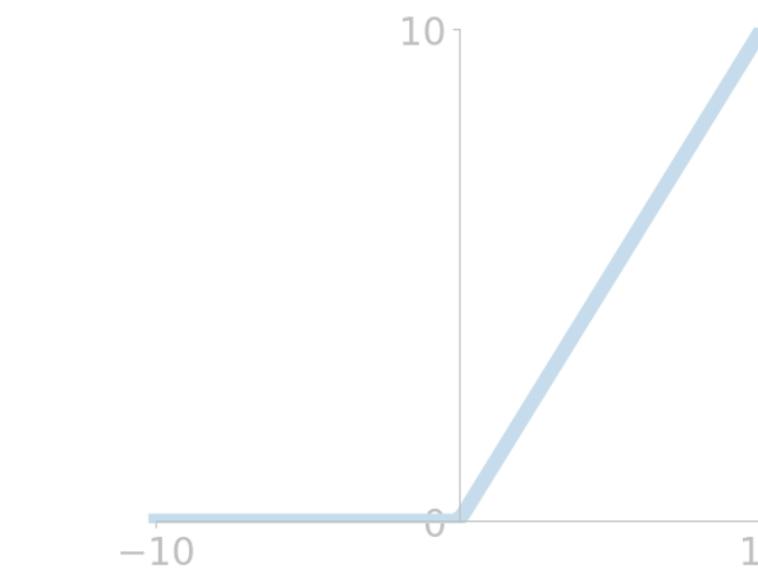
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Components of Convolutional Neural Networks

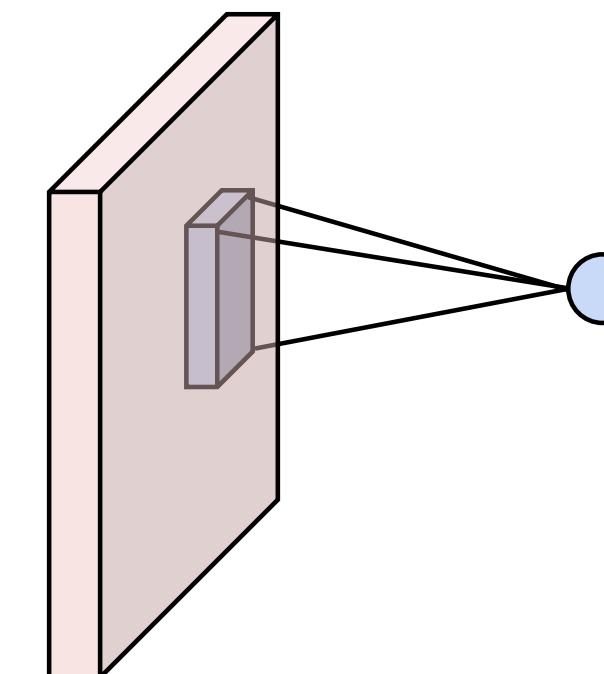
Fully-Connected Layers



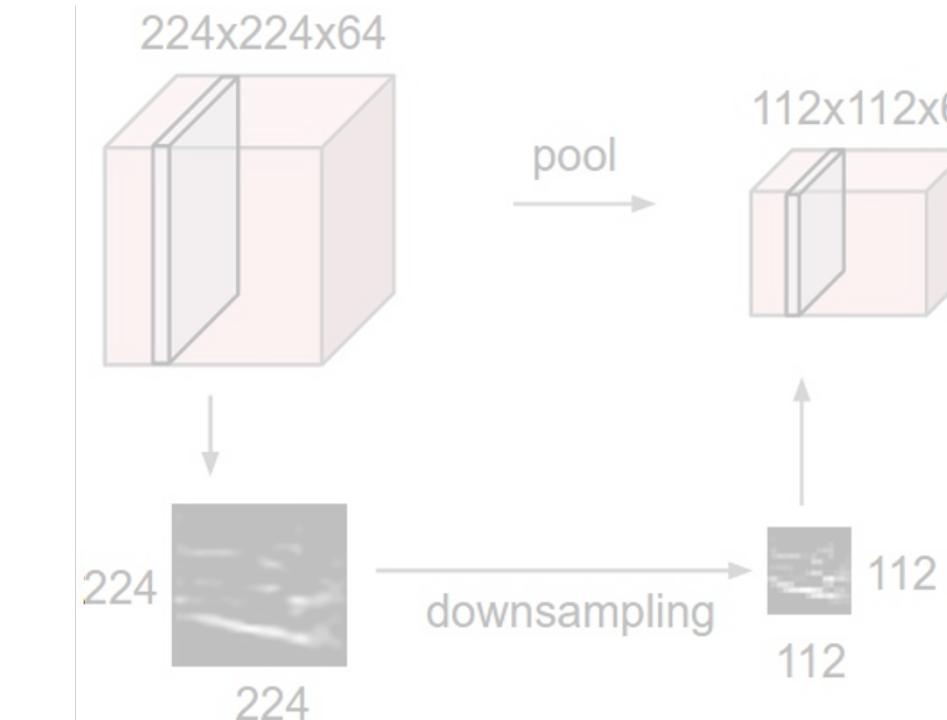
Activation Functions



Convolution Layers



Pooling Layers

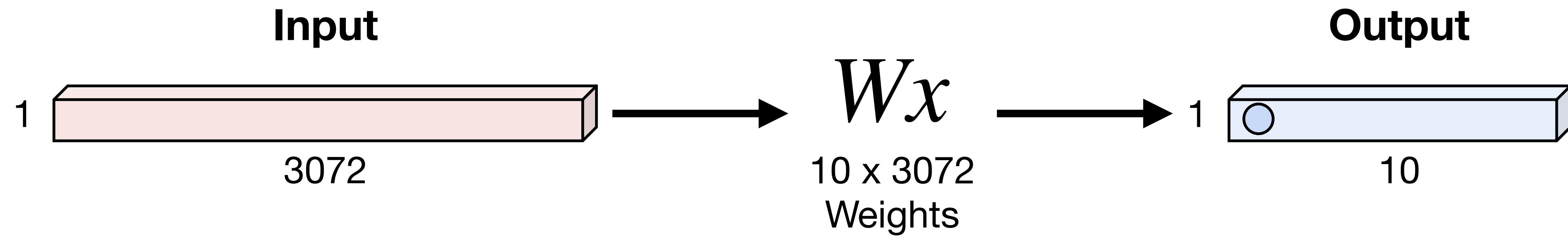


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

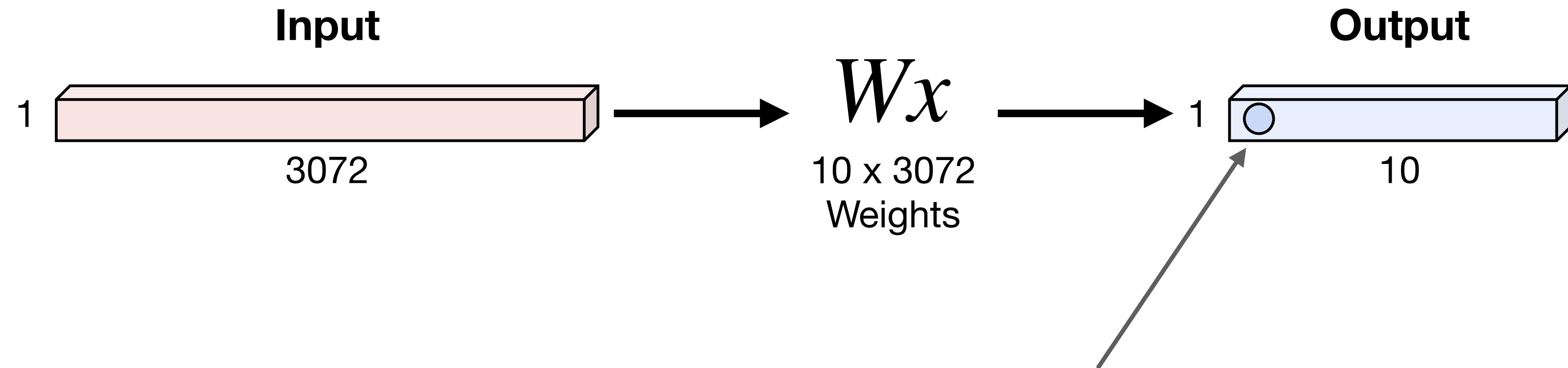
Fully-Connected Layer

3x32x32 image → stretch to 3072x1



Fully-Connected Layer

3x32x32 image → stretch to 3072x1

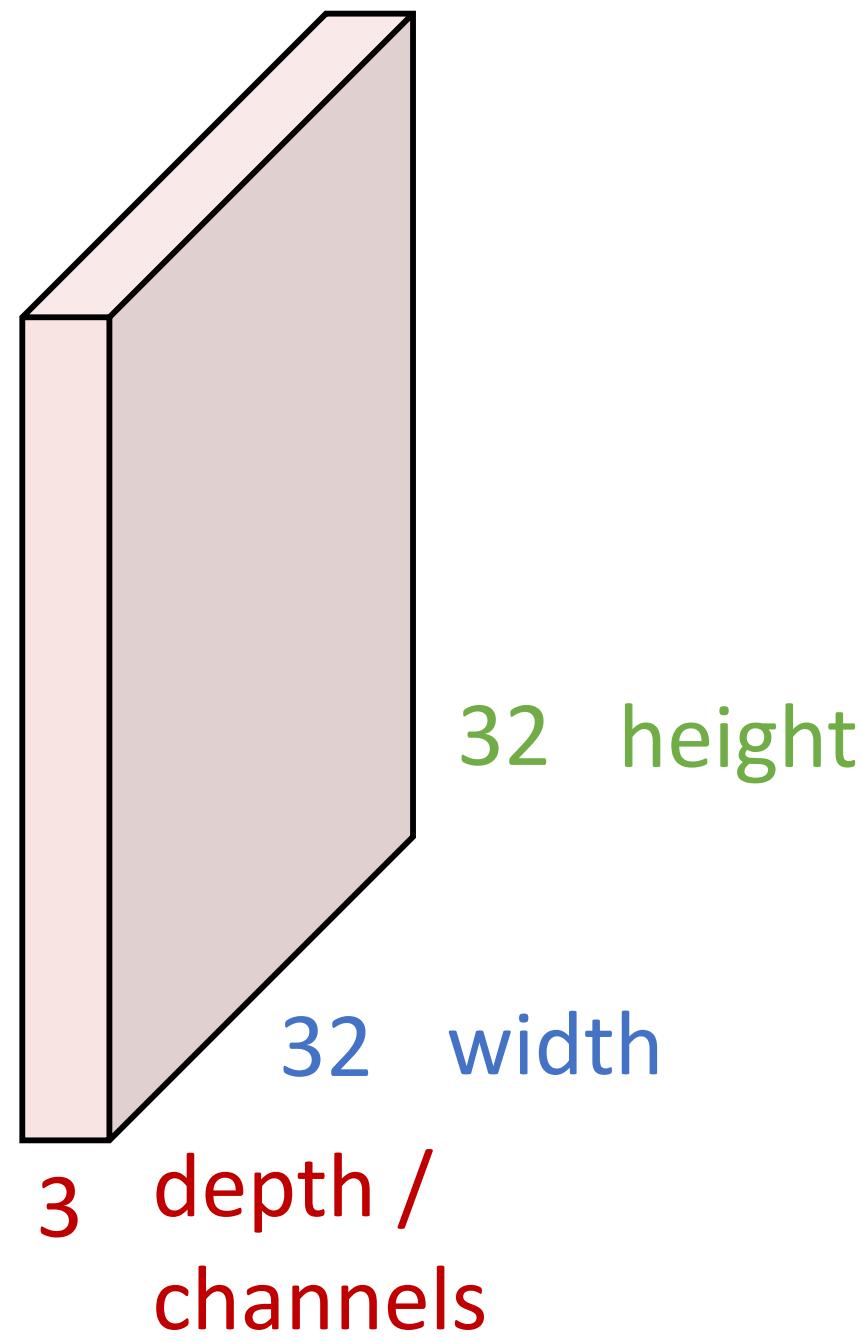


1 number:

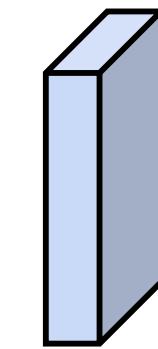
The result of taking a dot product
between a row of W and the input

Convolution Layer

$3 \times 32 \times 32$ image: preserve spatial structure



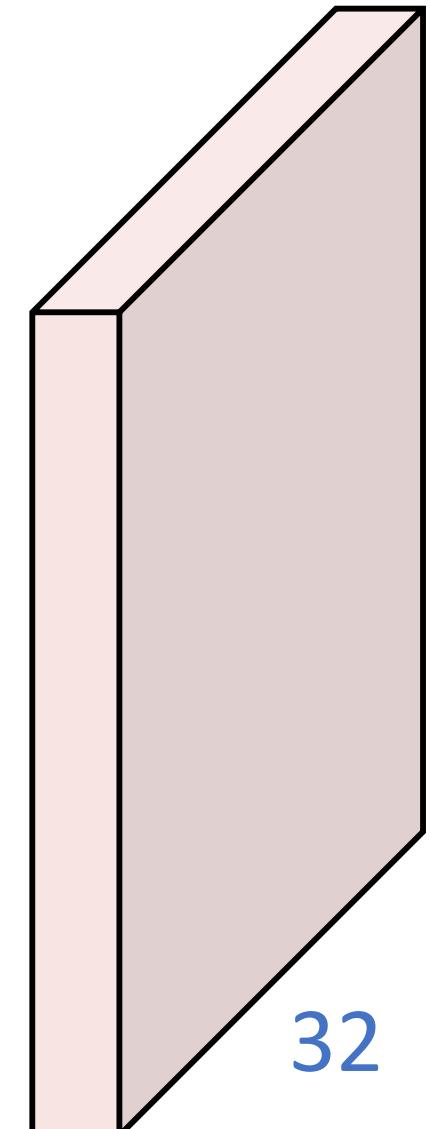
$3 \times 5 \times 5$ filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

$3 \times 32 \times 32$ image



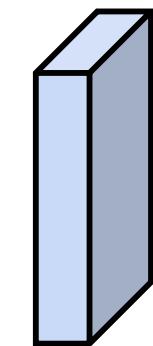
3 depth /
channels

32 width

32 height

Filters always extend the full depth
of the input volume

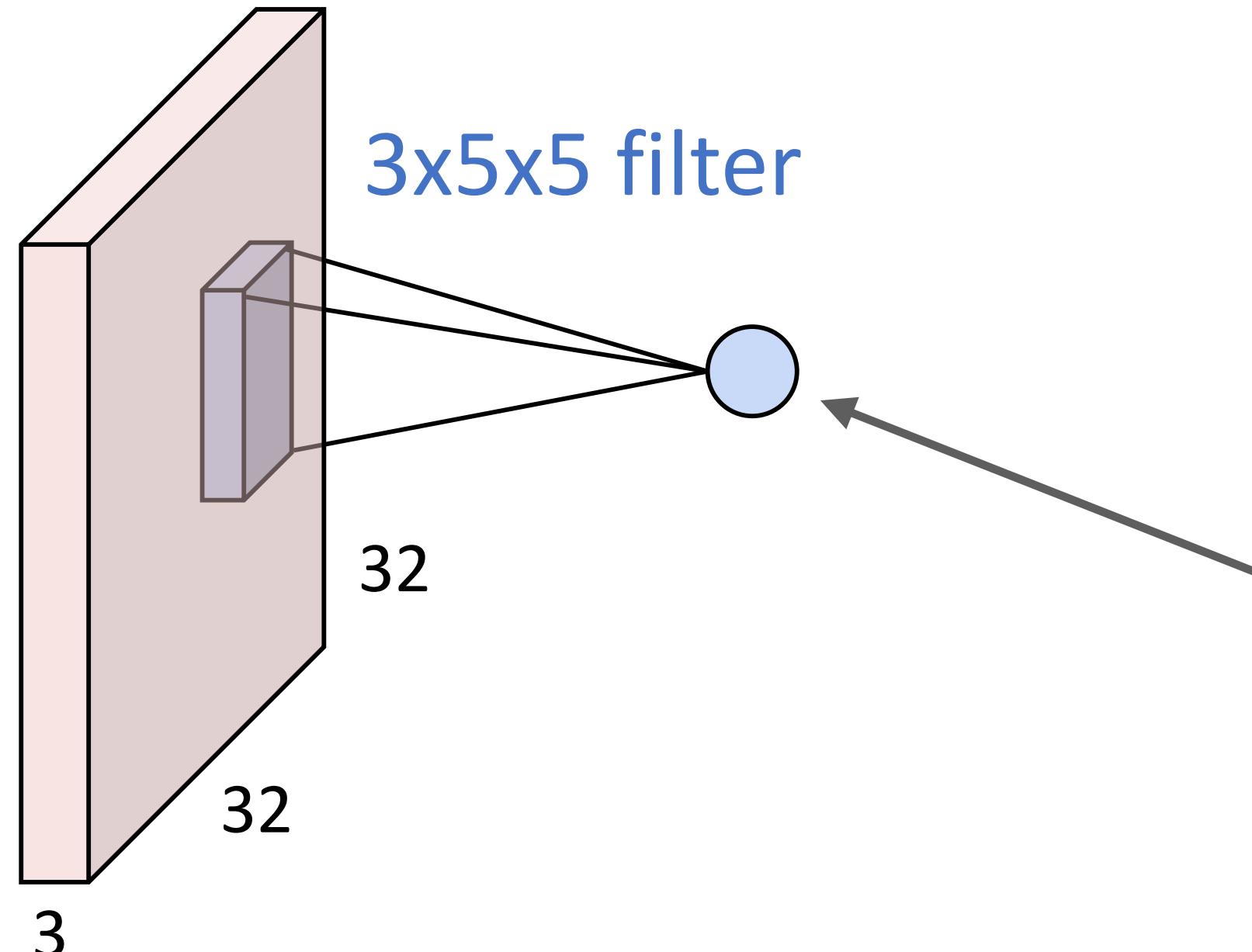
$3 \times 5 \times 5$ filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

3x32x32 image



1 number:

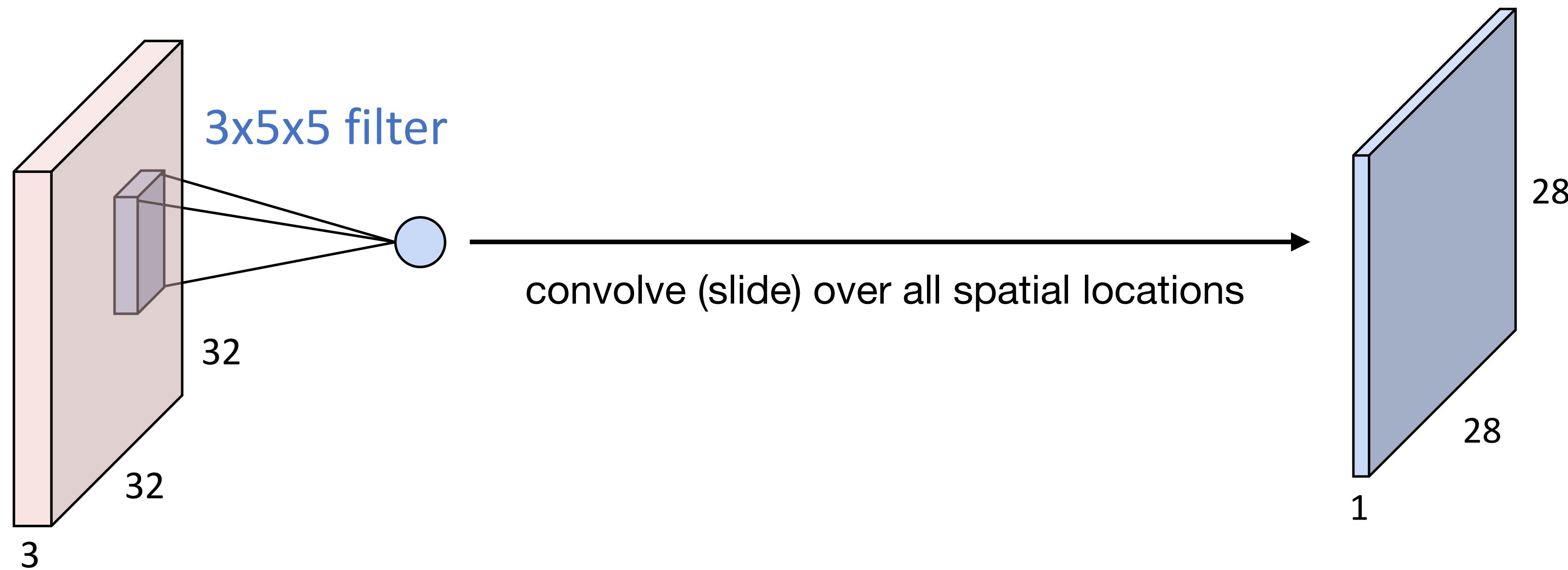
The result of taking a dot product between the filter and a small $3 \times 5 \times 5$ portion of the image
(i.e. $3 \times 5 \times 5 = 75$ -dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

3x32x32 image

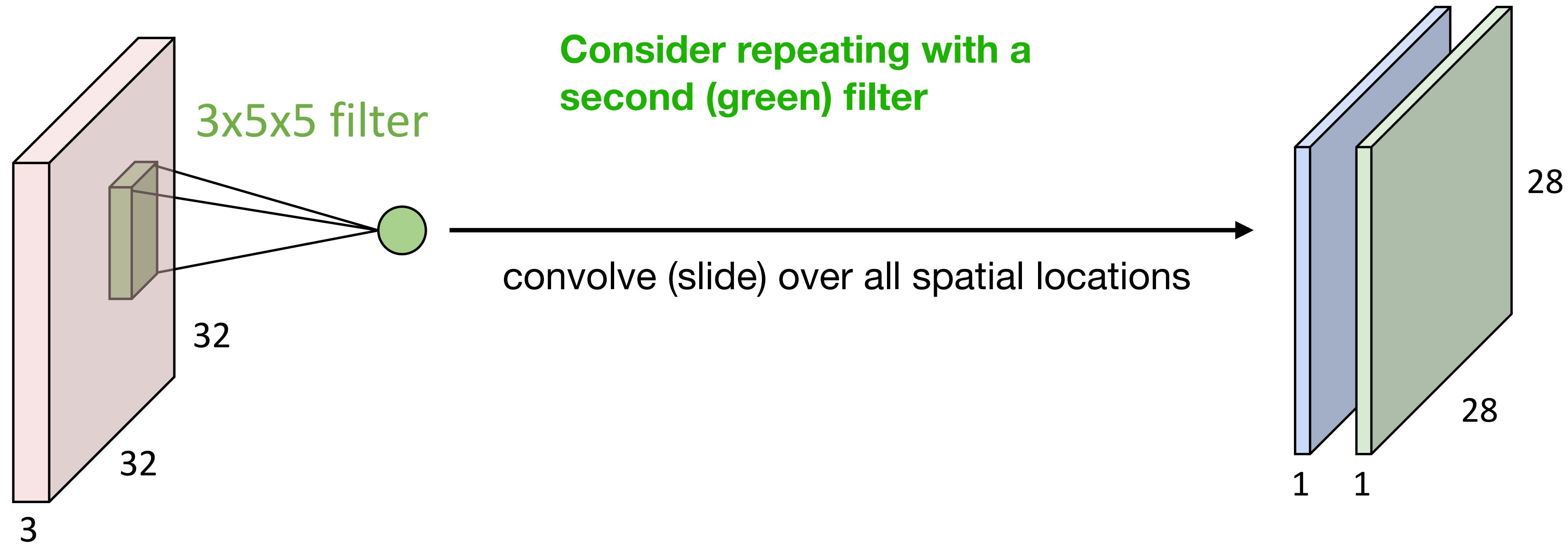
1x28x28 activation map



Convolution Layer

3x32x32 image

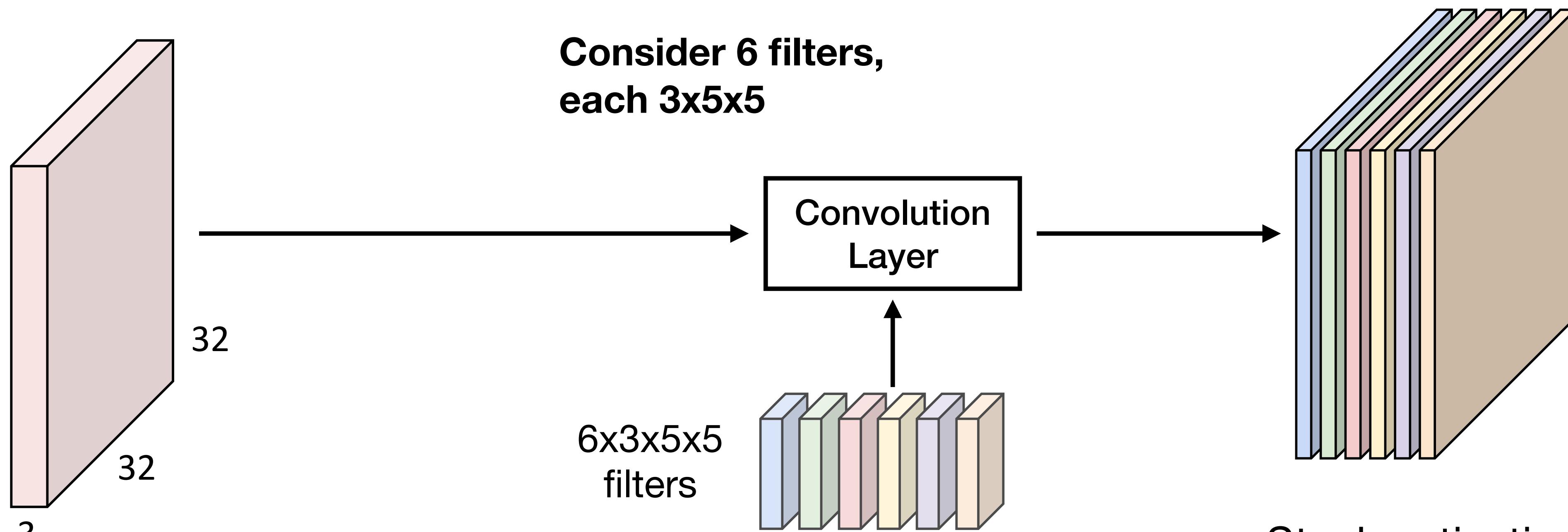
two 1x28x28 activation map



Convolution Layer

3x32x32 image

six 1x28x28 activation map

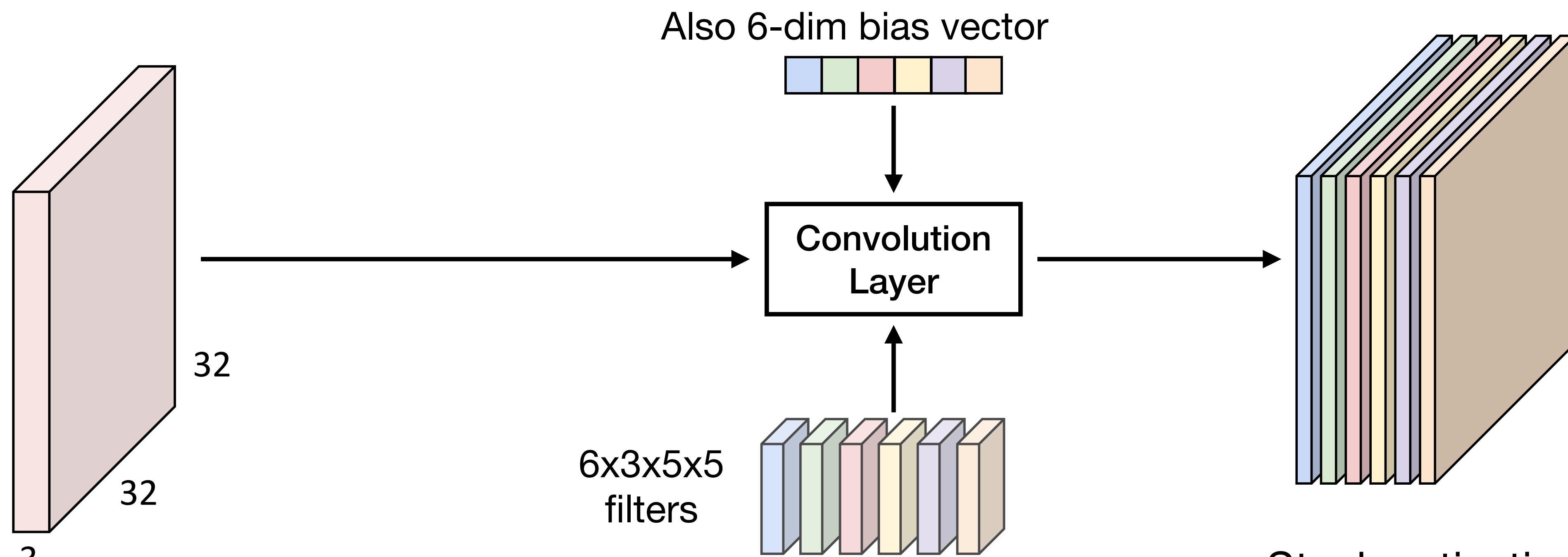


Stack activations to get
a 6x28x28 output image

Convolution Layer

3x32x32 image

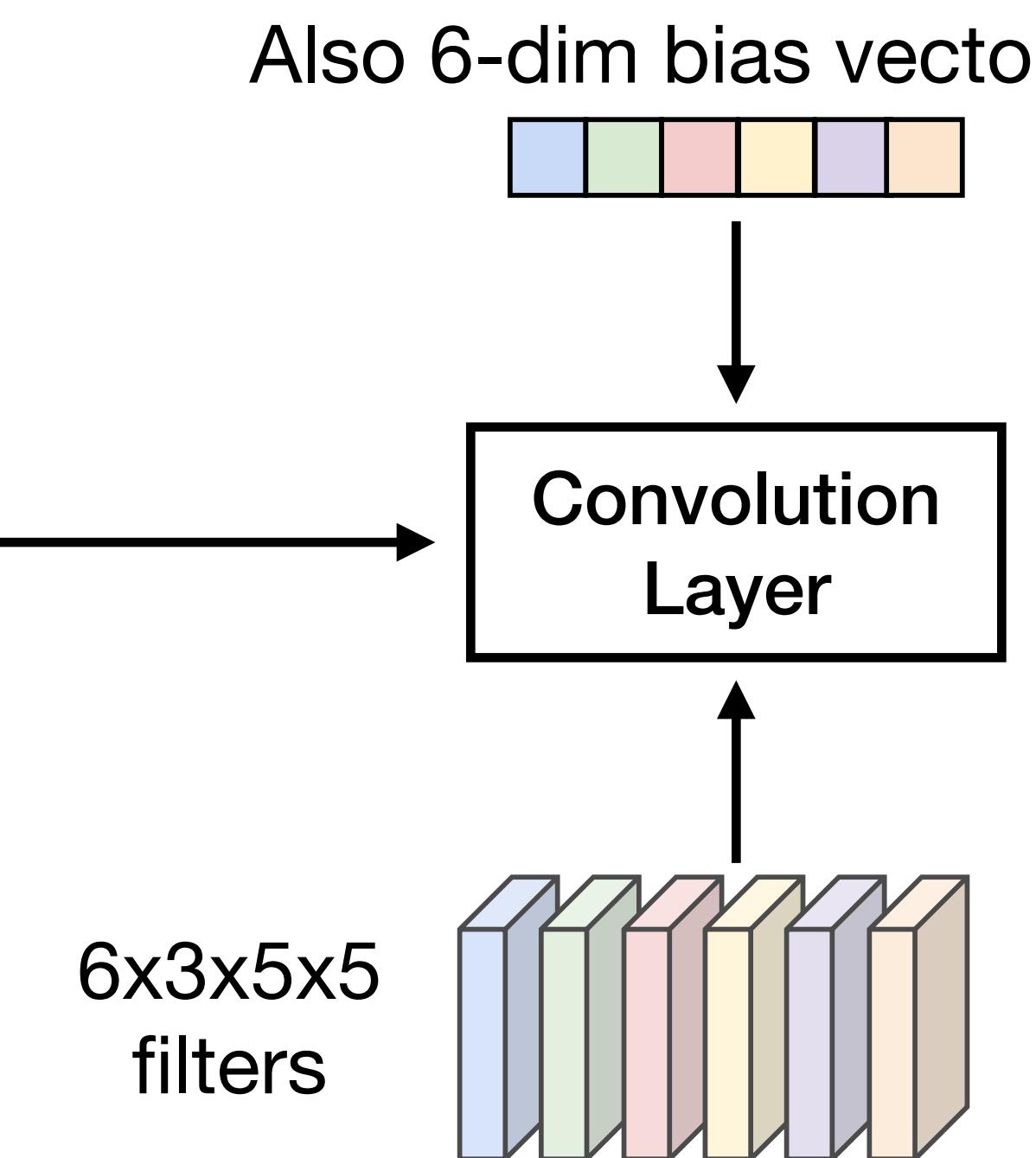
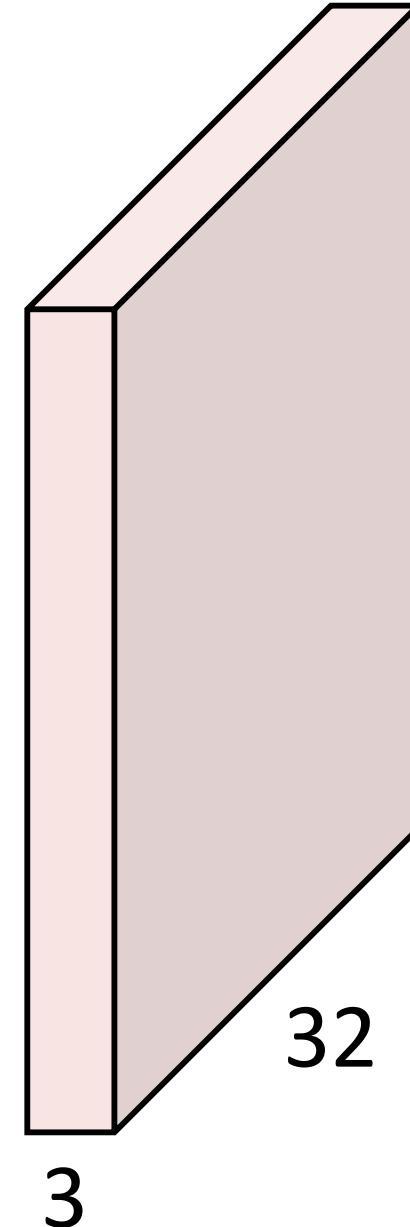
six 1x28x28 activation map



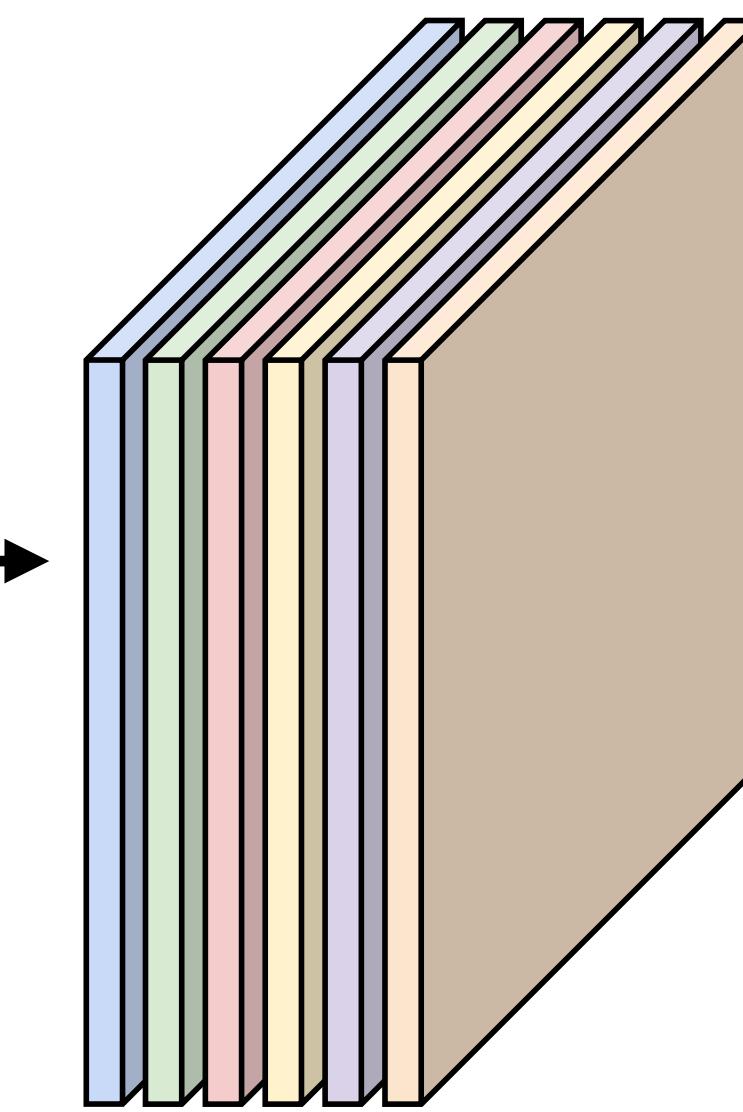
Stack activations to get
a 6x28x28 output image

Convolution Layer

3x32x32 image

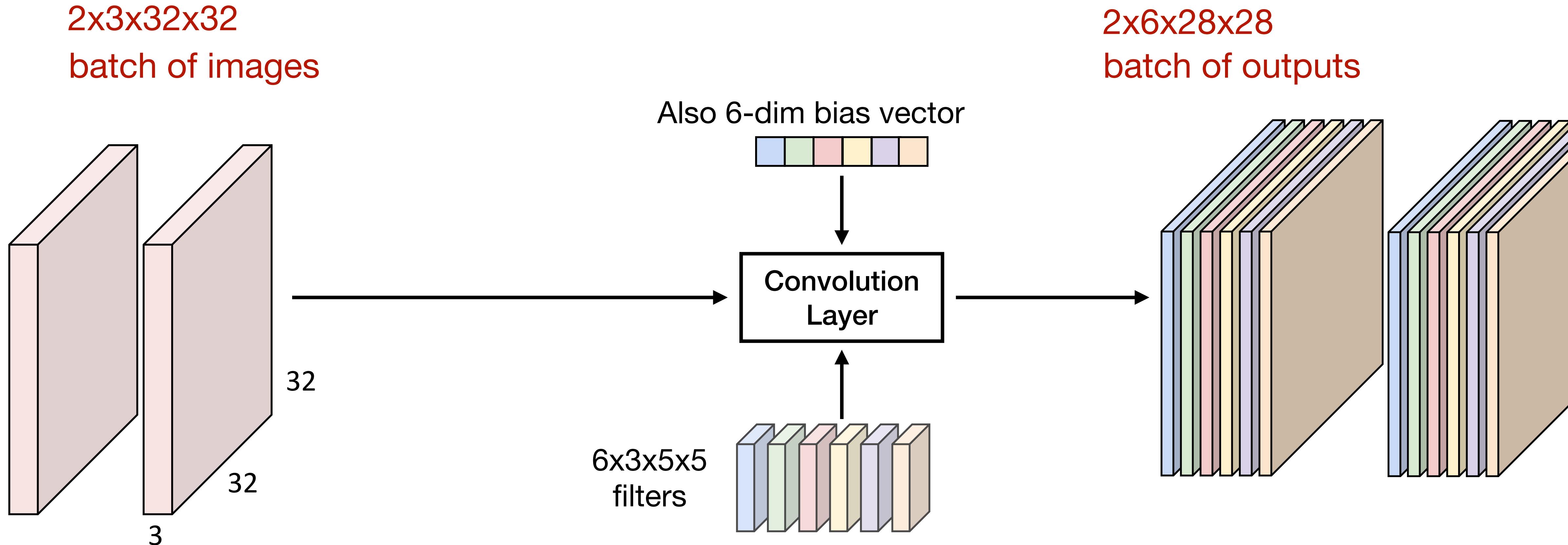


28x28 grid, at each point a 6-dim vector

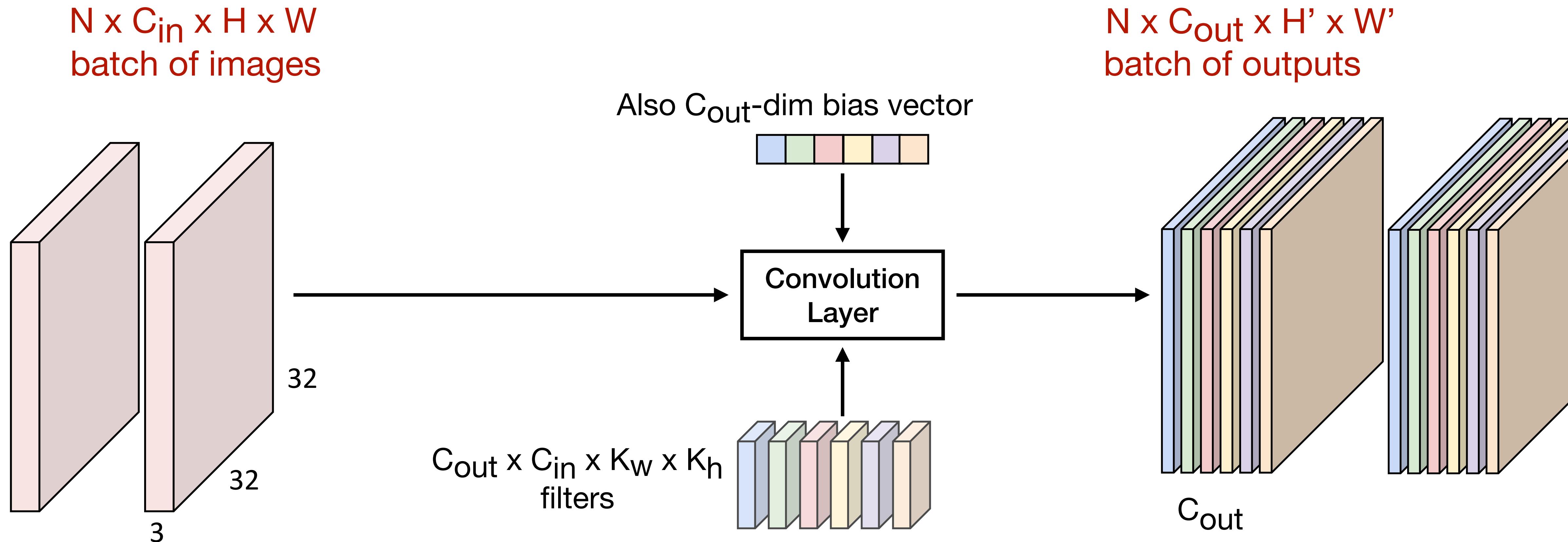


Stack activations to get a 6x28x28 output image

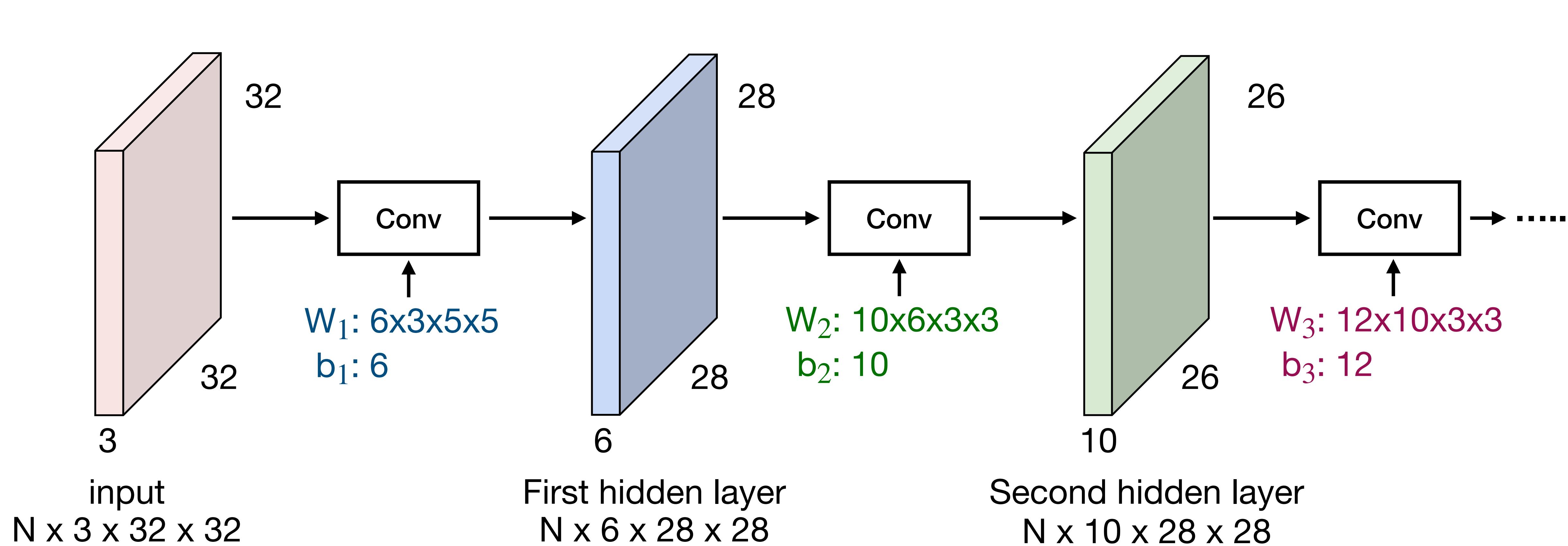
Convolution Layer



Convolution Layer

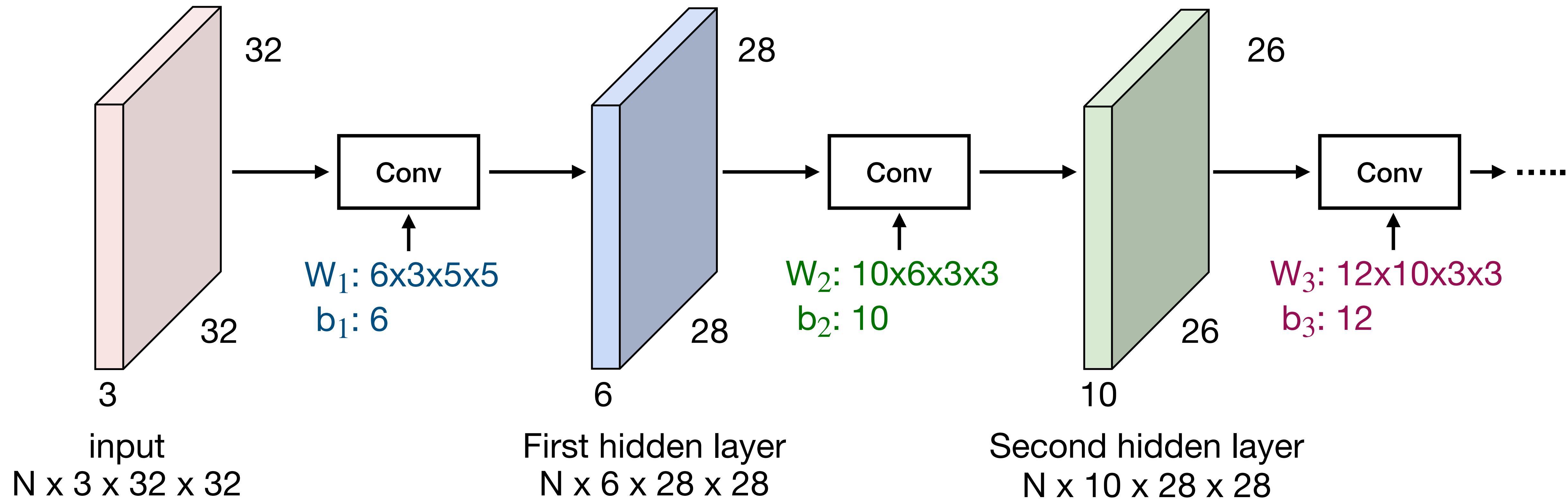


Stacking Convolutions



Stacking Convolutions

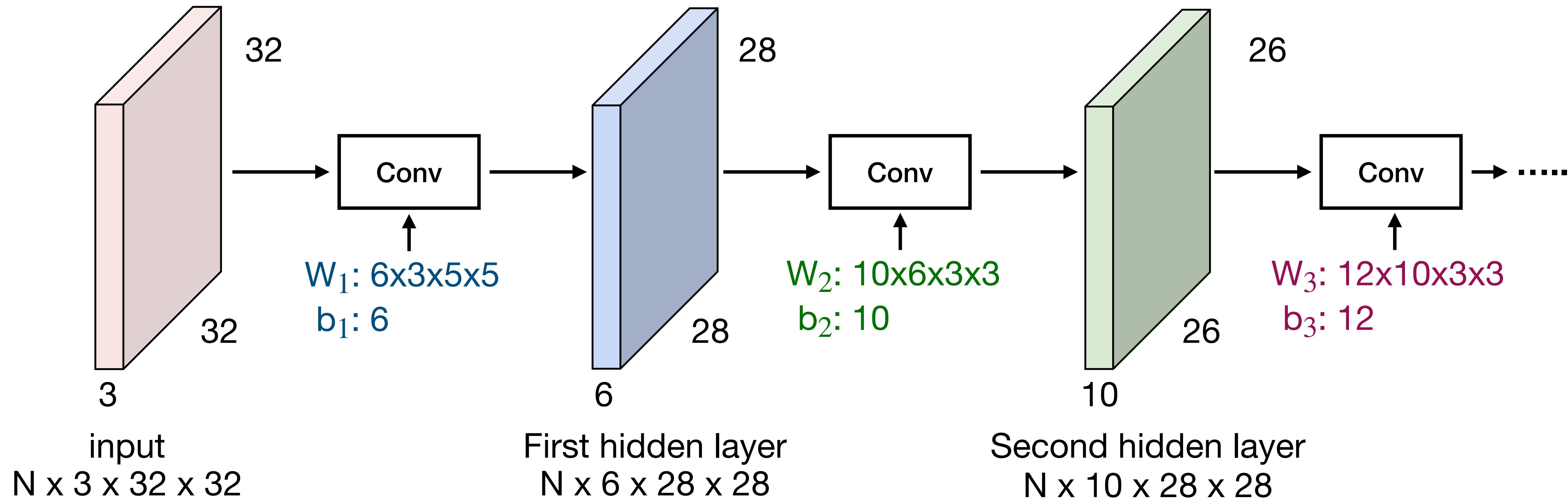
Q: What happens if we stack two convolution layers?



Stacking Convolutions

Q: What happens if we stack two convolution layers?

(Recall $y=W_2W_1x$ is a linear classifier)

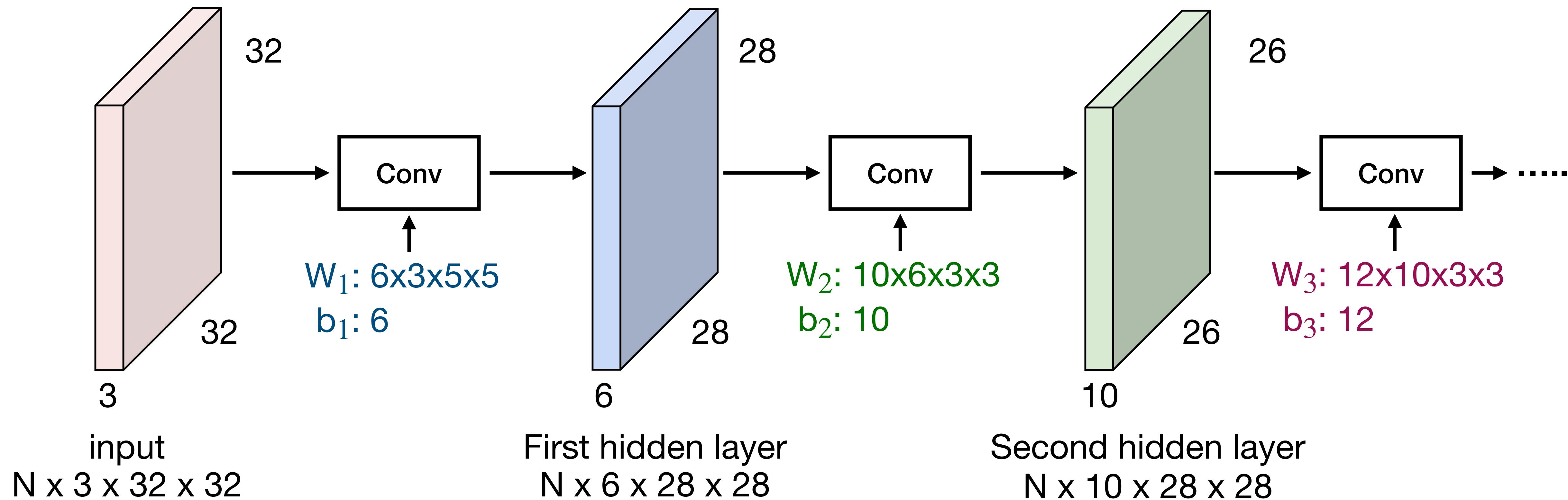


Stacking Convolutions

Q: What happens if we stack two convolution layers?

(Recall $y=W_2W_1x$ is a linear classifier)

A: We get another convolution!

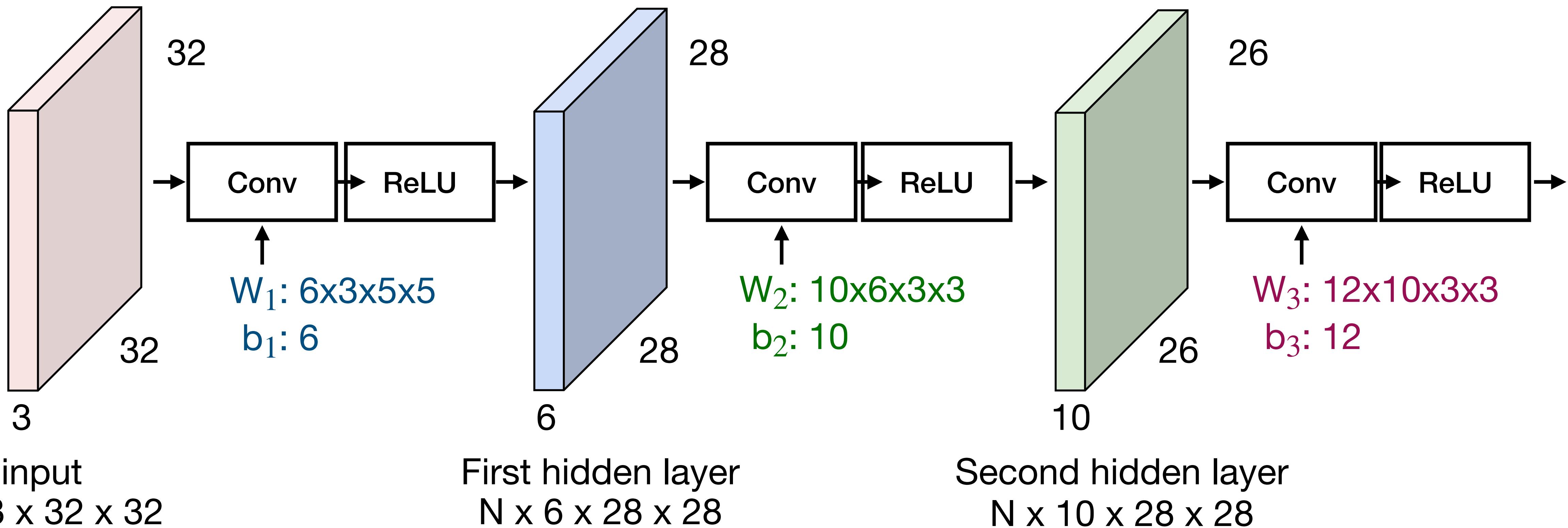


Stacking Convolutions

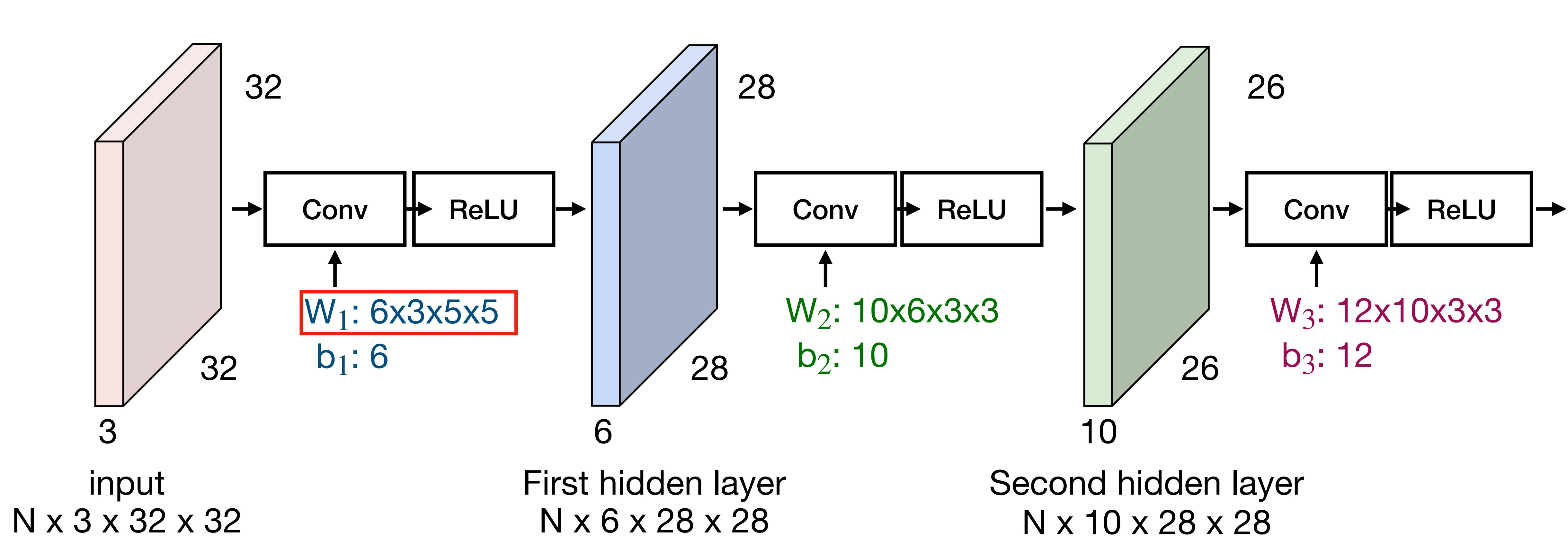
Q: What happens if we stack two convolution layers?

(Recall $y=W_2W_1x$ is a linear classifier)

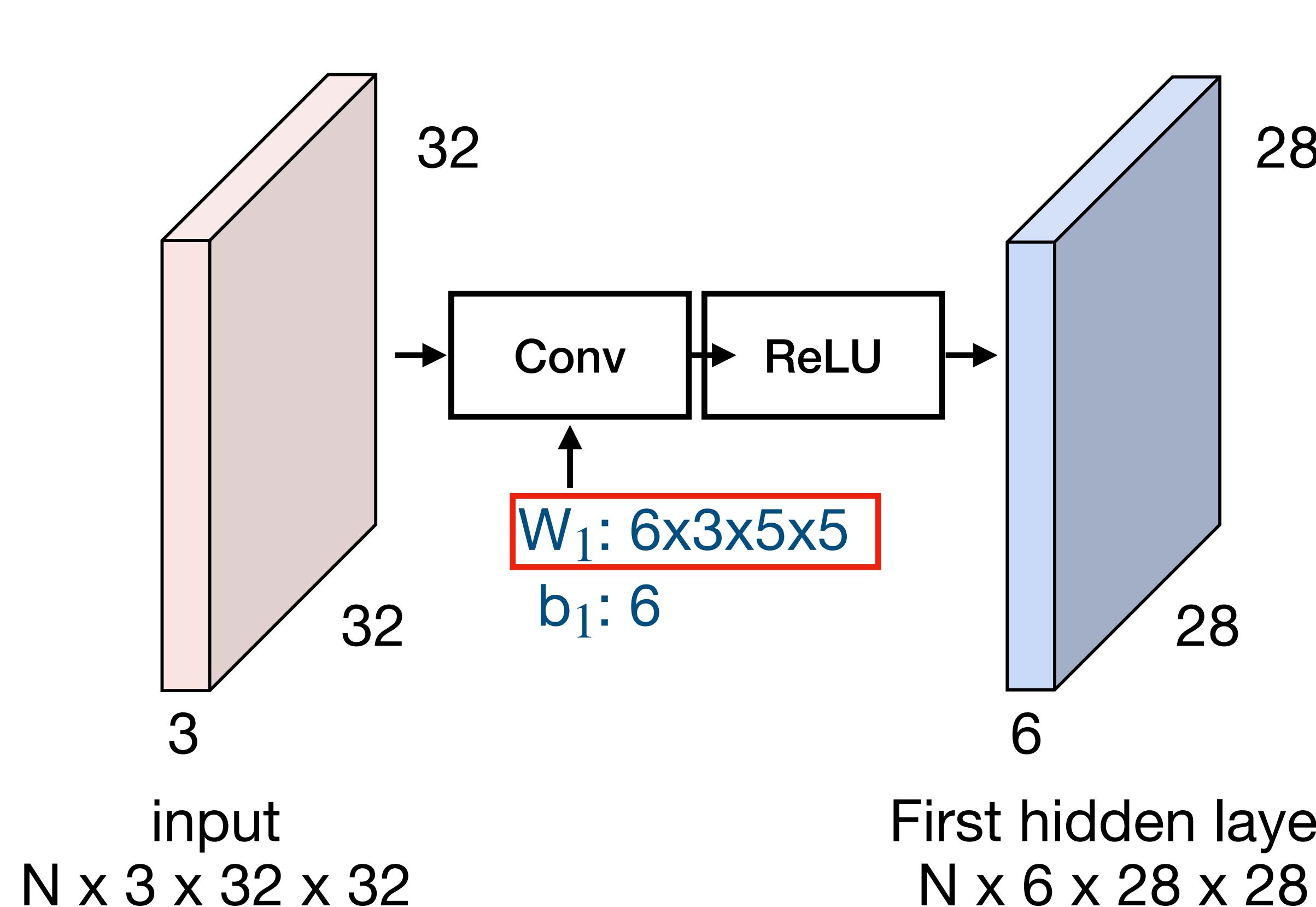
A: We get another convolution!



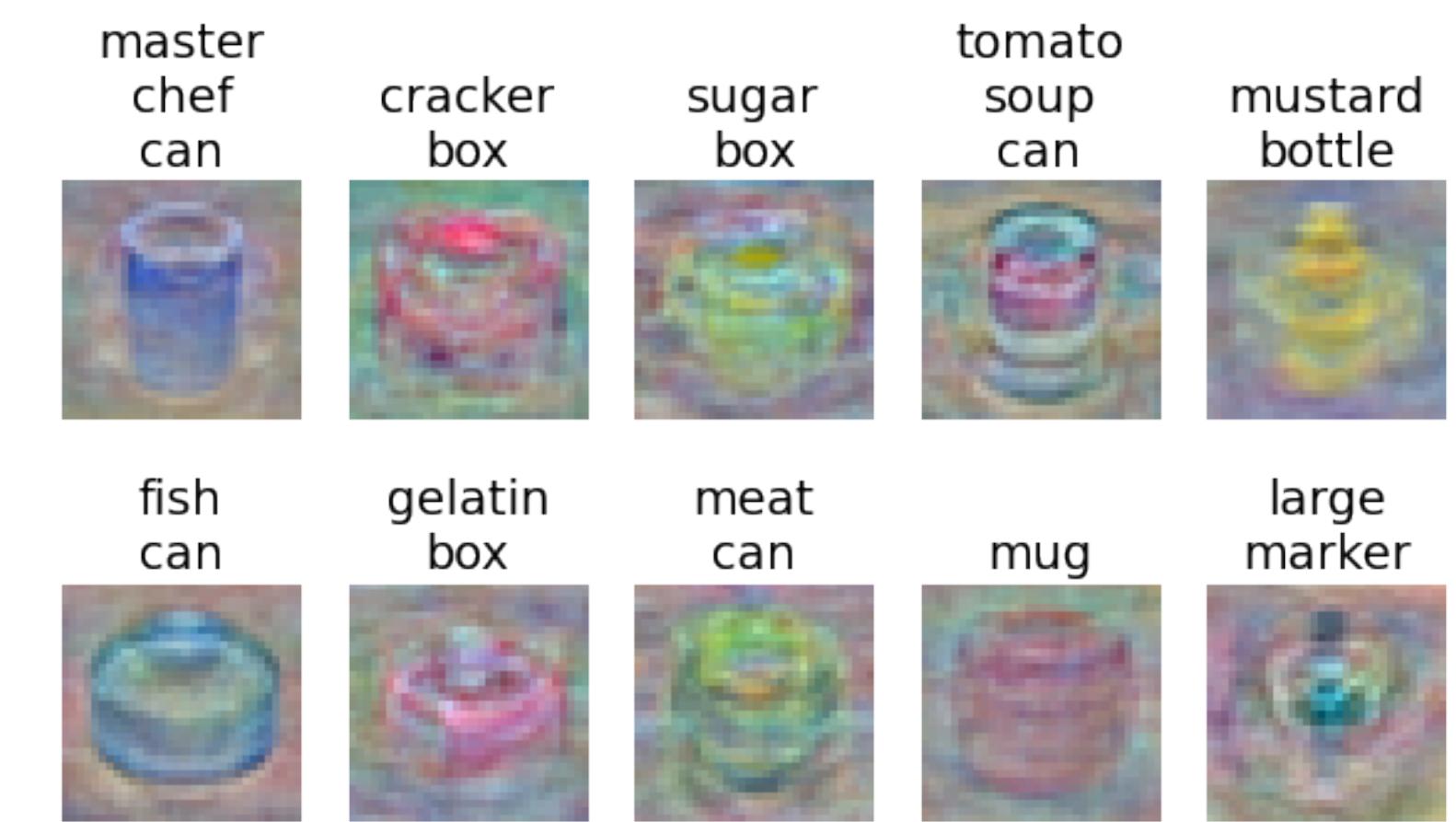
What do convolutional filters learn?



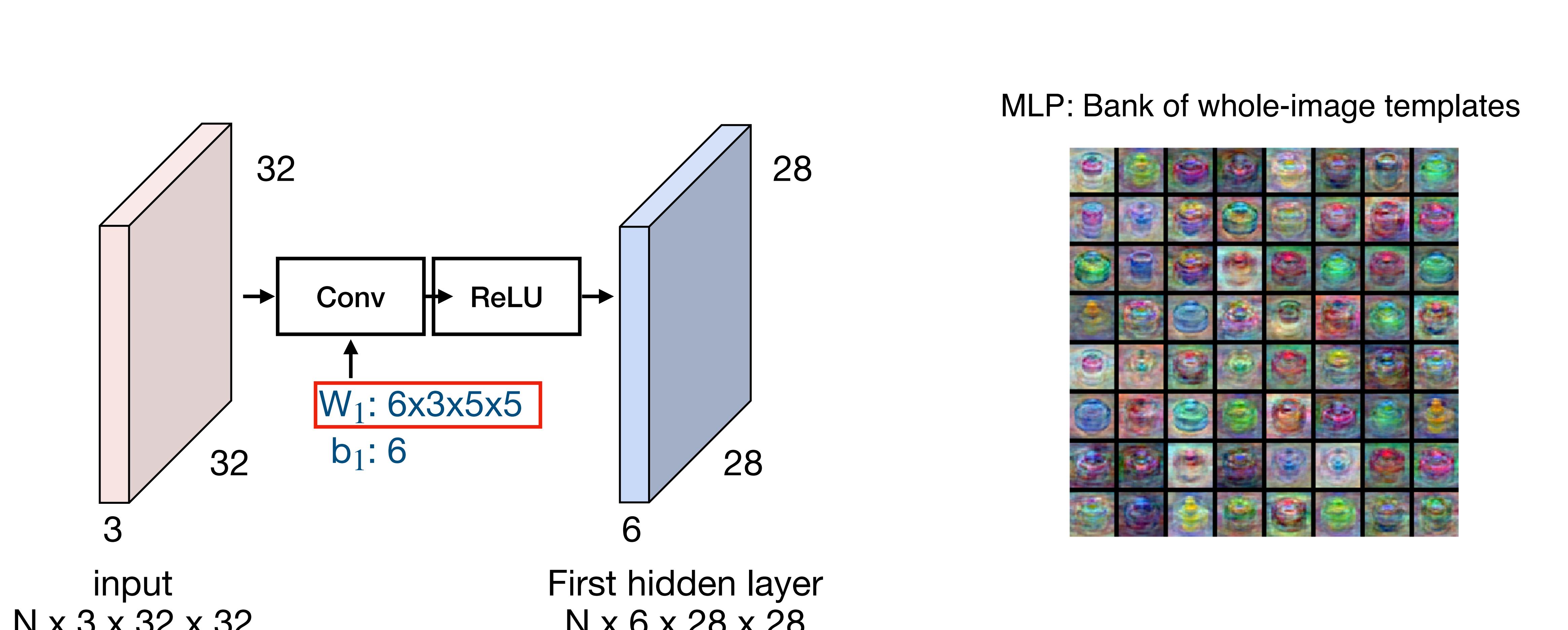
What do convolutional filters learn?



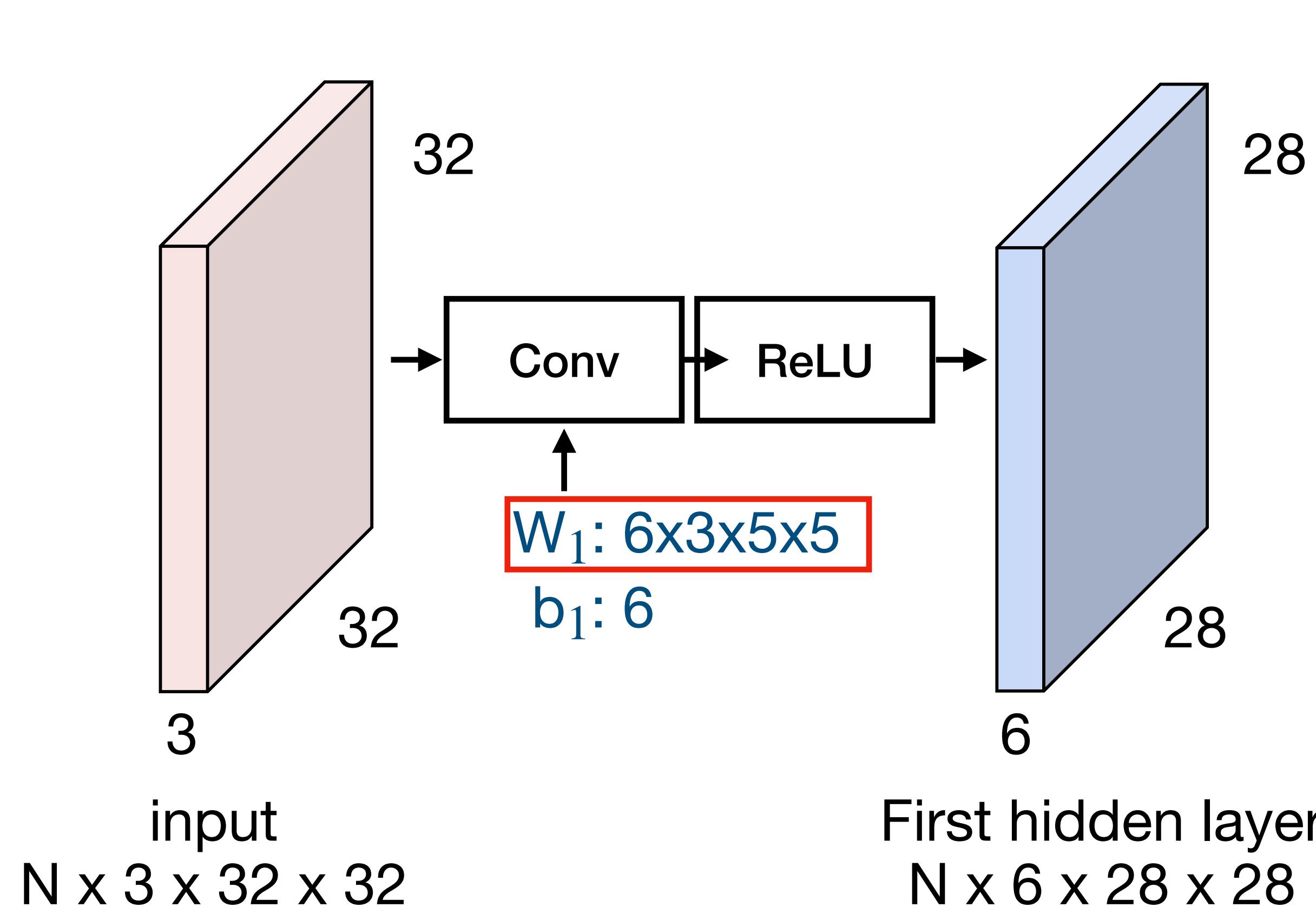
Linear classifier: One template per class



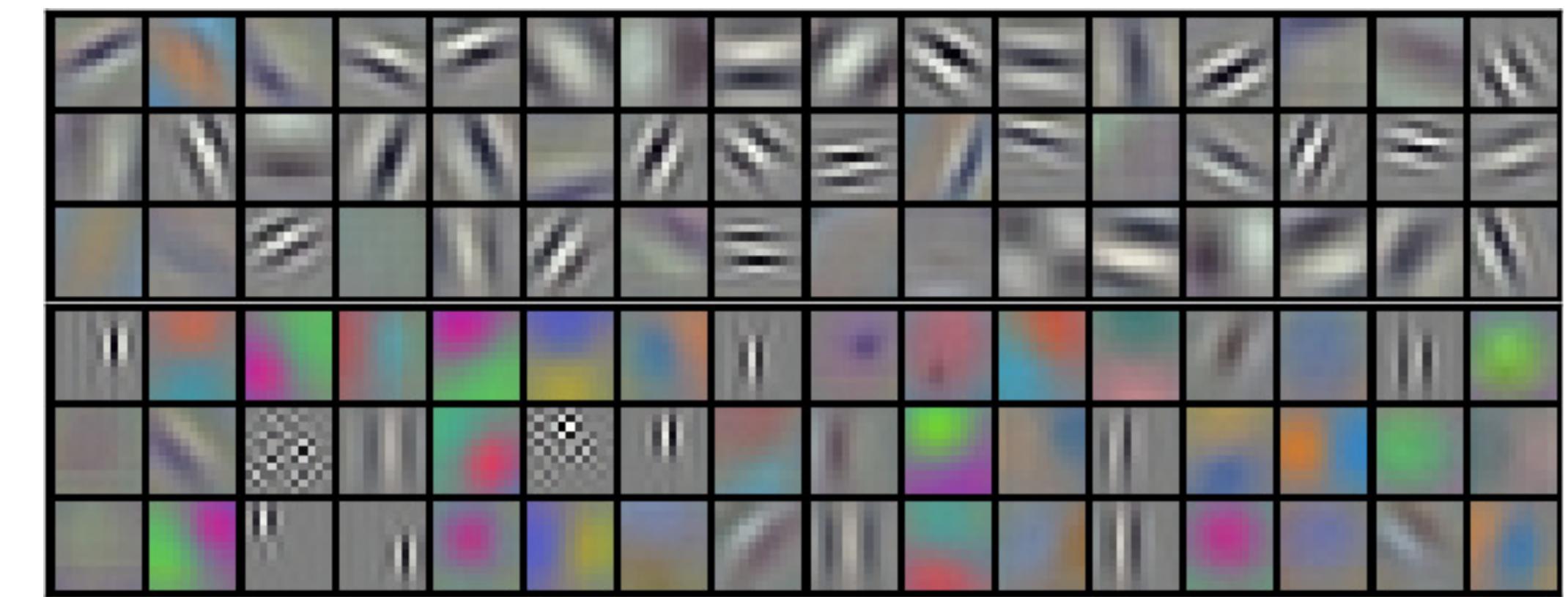
What do convolutional filters learn?



What do convolutional filters learn?

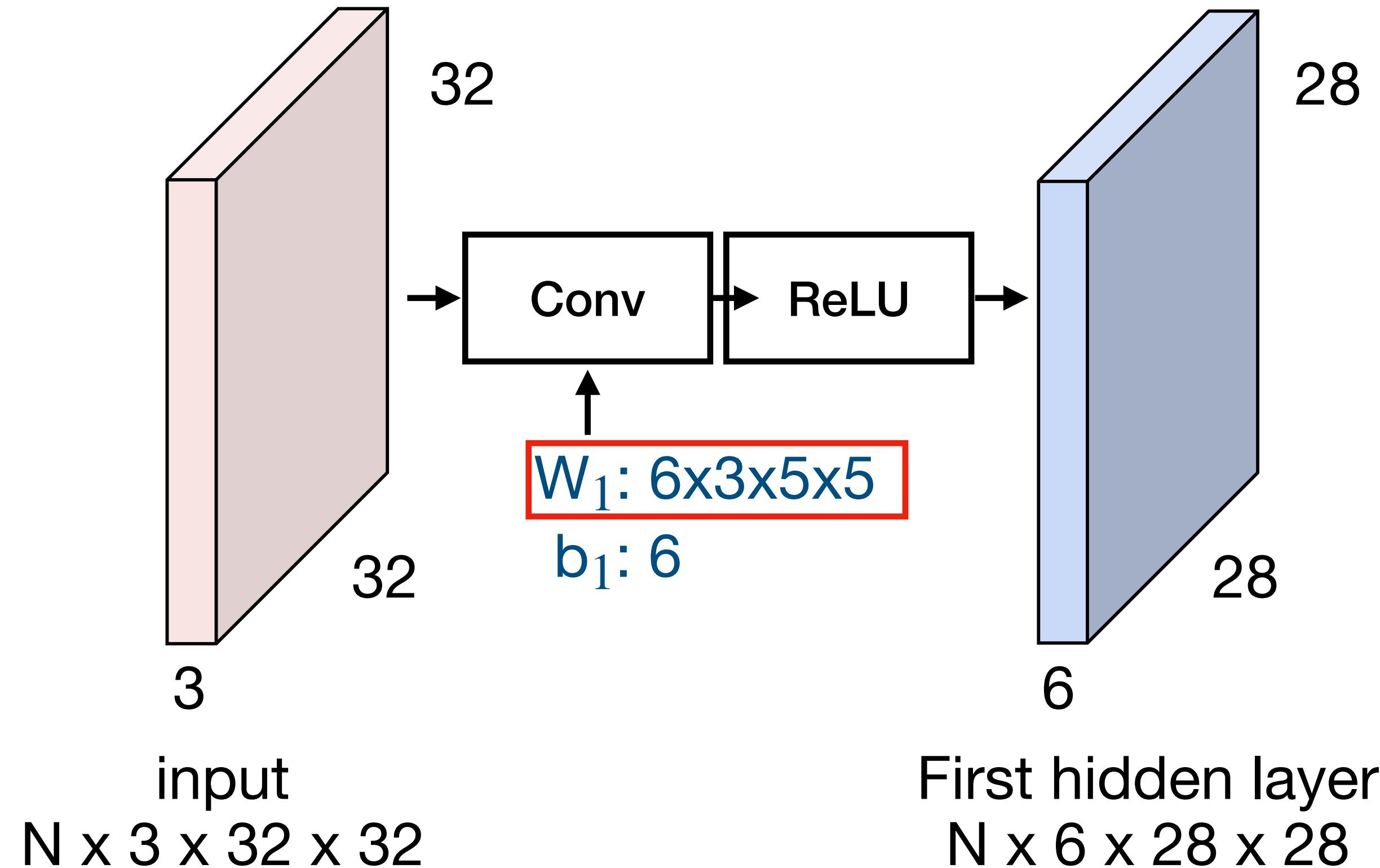


First-layer conv filters: local image templates
(often learns oriented edges, opposing colors)

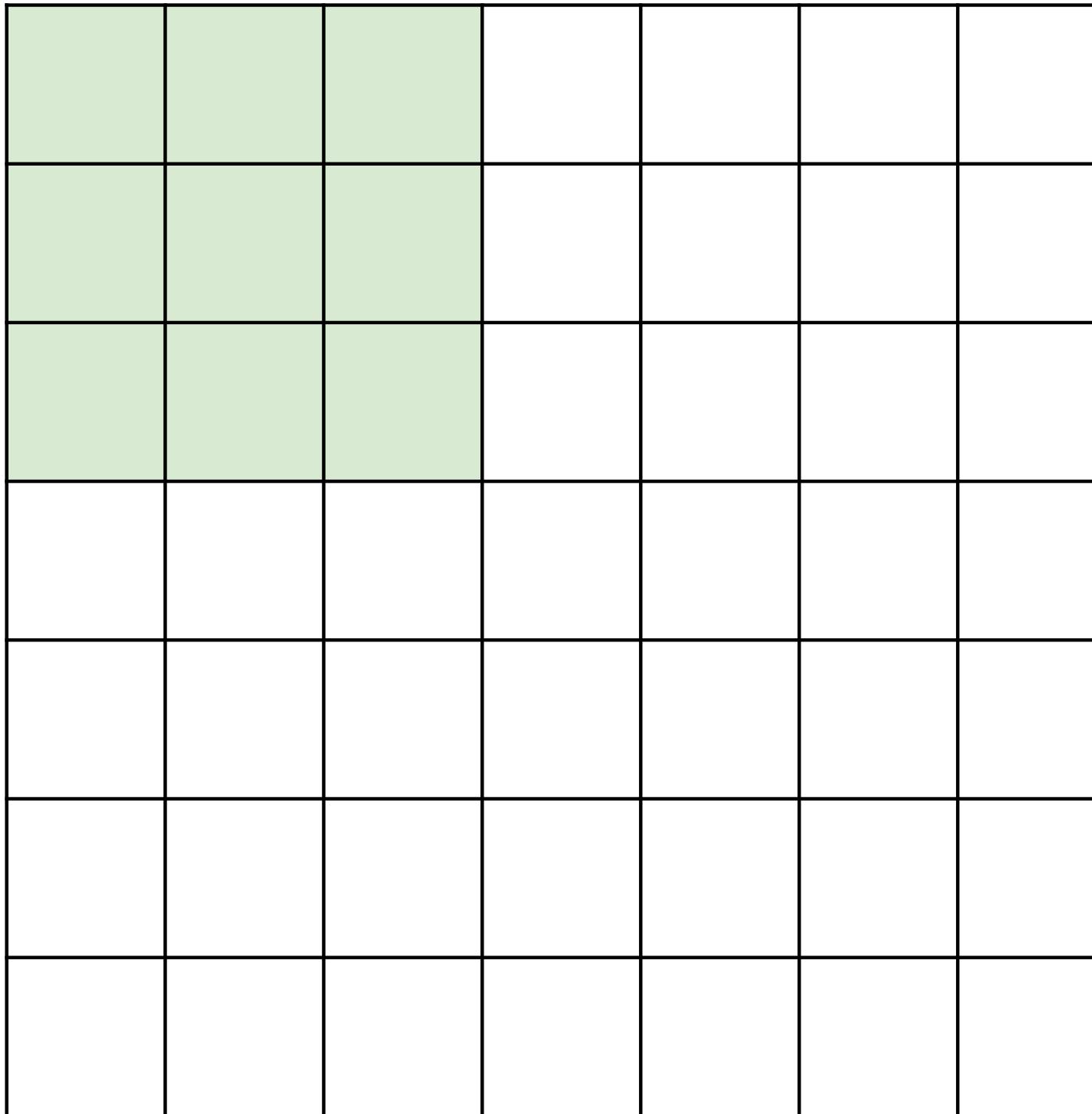


[AlexNet](#): 96 filters, each $3 \times 11 \times 11$

A closer look at spatial dimensions



A closer look at spatial dimensions

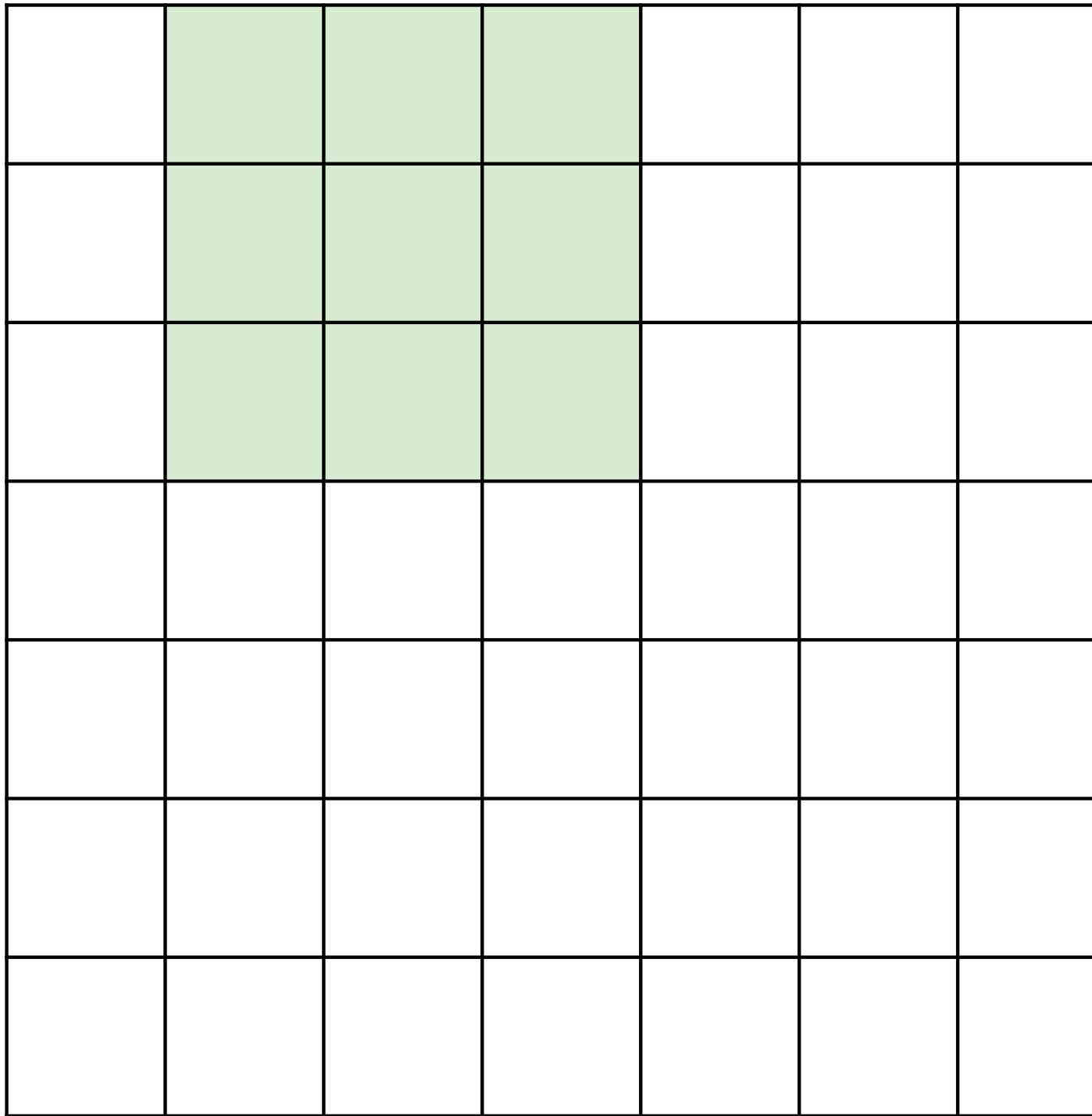


7

Input: 7x7
Filter: 3x3

7

A closer look at spatial dimensions

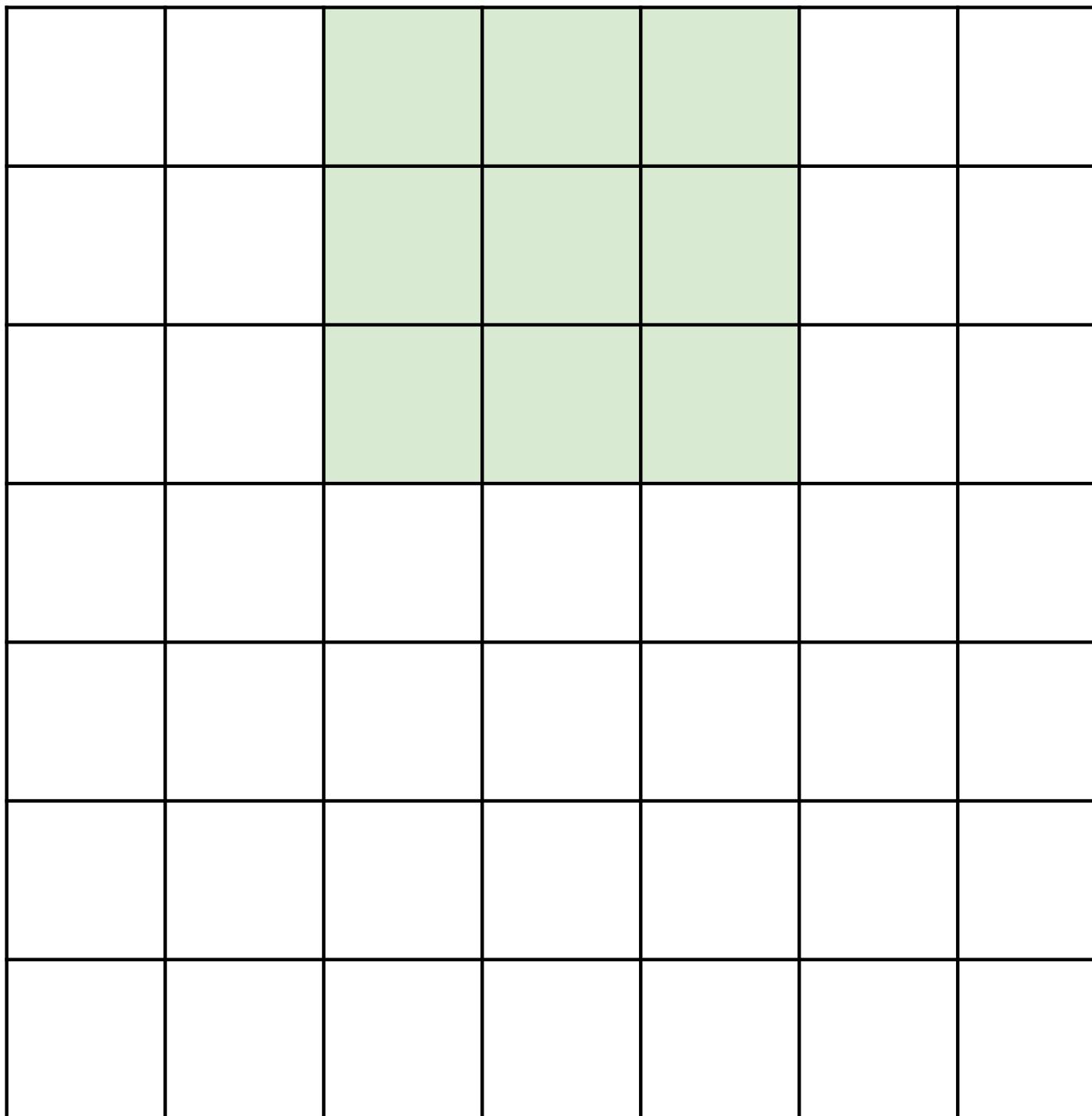


7

Input: 7x7
Filter: 3x3

7

A closer look at spatial dimensions

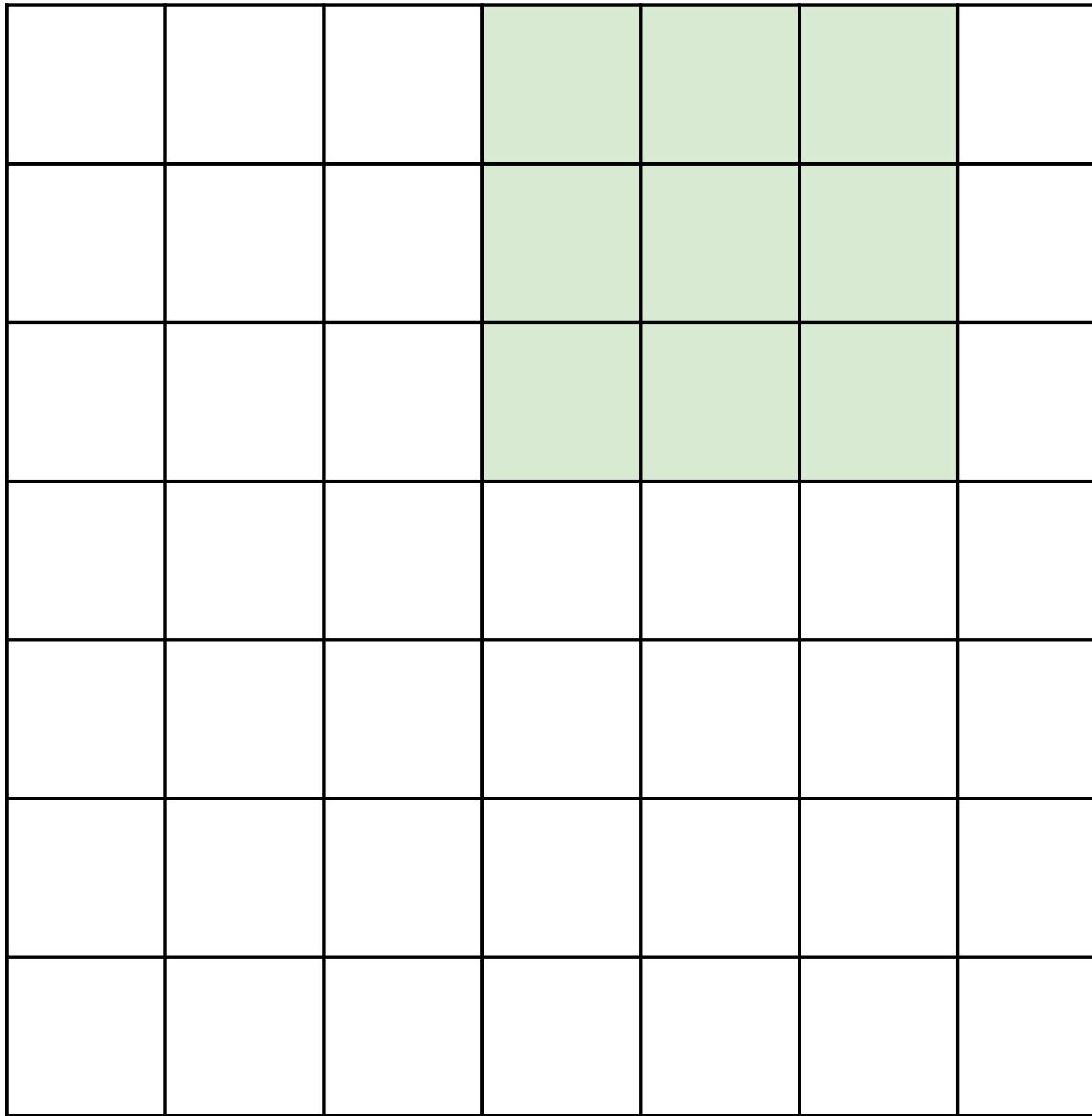


7

Input: 7x7
Filter: 3x3

7

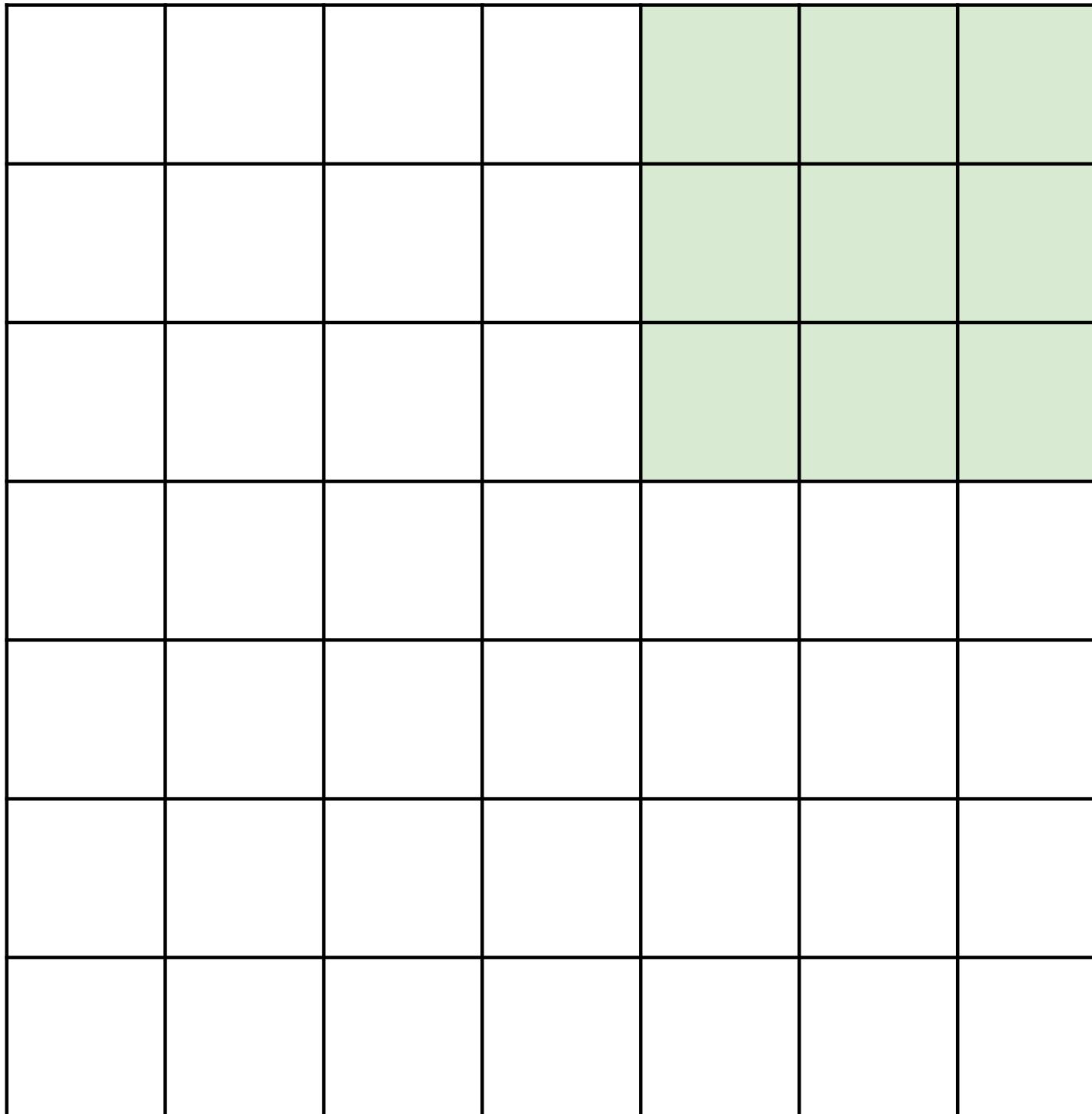
A closer look at spatial dimensions



7

Input: 7x7
Filter: 3x3

A closer look at spatial dimensions

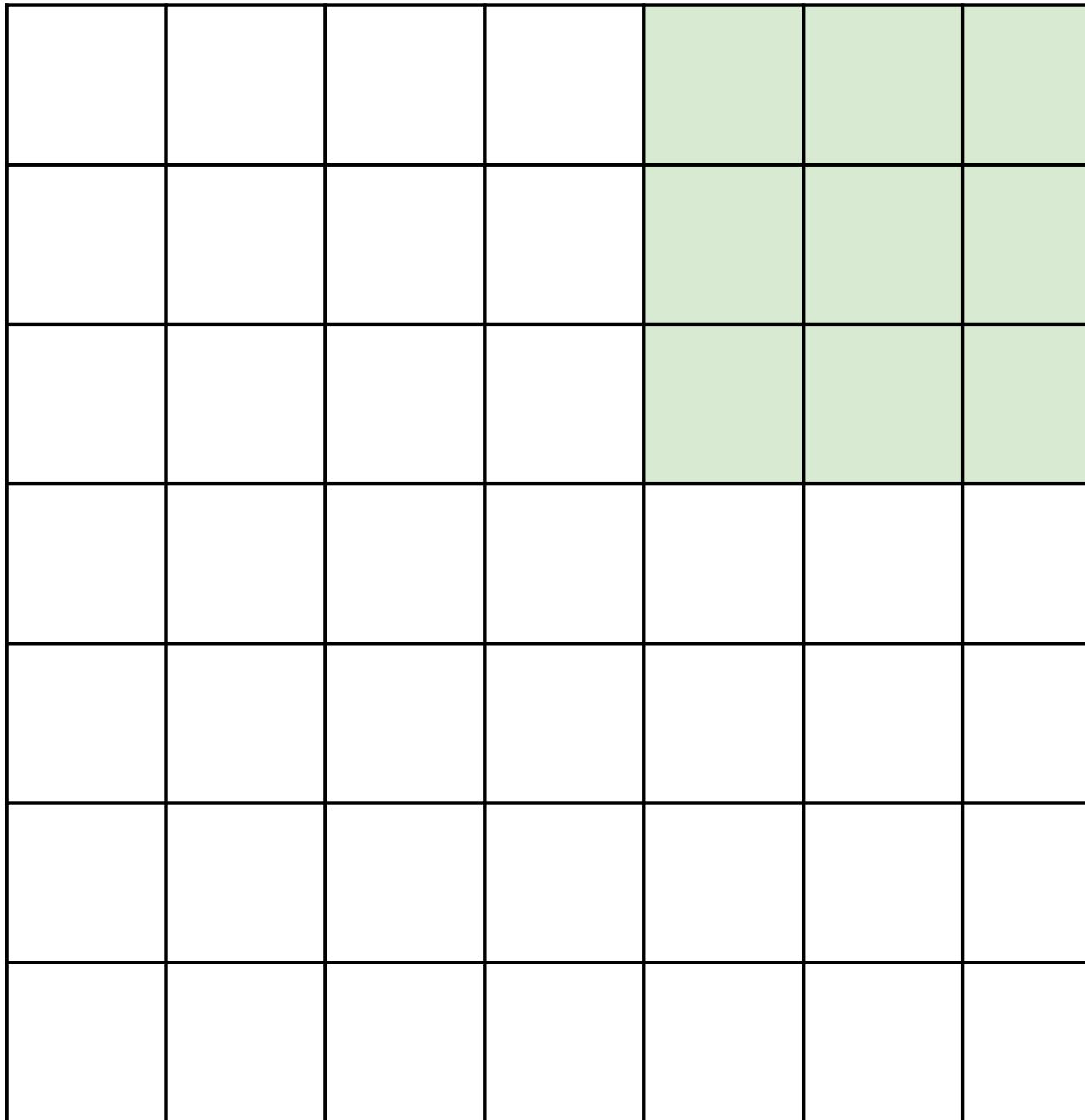


7

Input: 7x7
Filter: 3x3
Output: 5x5

7

A closer look at spatial dimensions



7

7

Input: 7x7

Filter: 3x3

Output: 5x5

In general:

Input: W

Filter: K

Output: $W - K + 1$

Problem: Feature
maps “shrink”
with each layer!

A closer look at spatial dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input: 7x7

Filter: 3x3

Output: 5x5

In general:

Input: W

Filter: K

Output: $W - K + 1$

Problem: Feature
maps “shrink”
with each layer!

Solution: padding

Add zeros around the input

A closer look at spatial dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input: 7x7

Filter: 3x3

Output: 5x5

In general:

Input: W

Filter: K

Padding: P

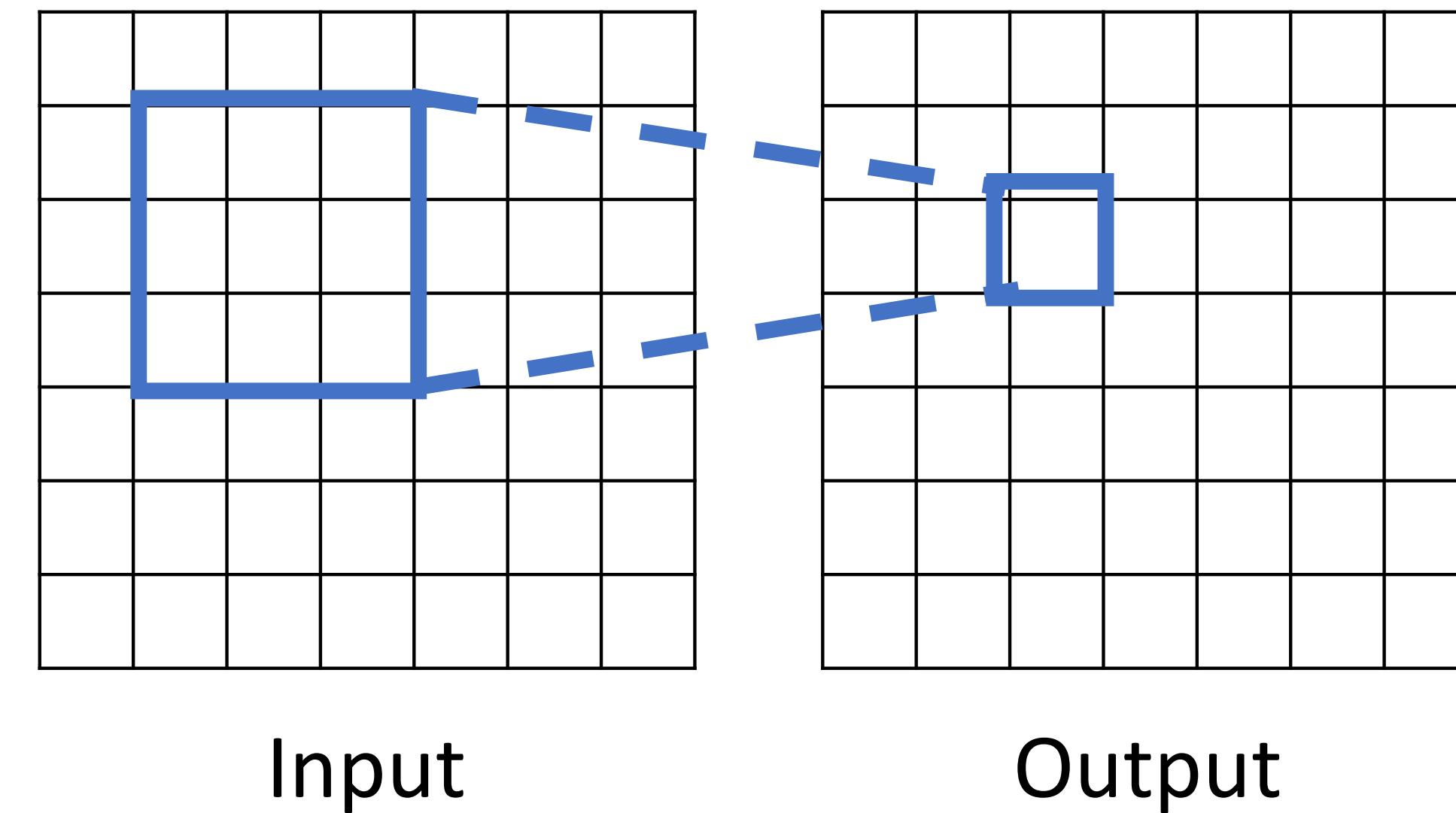
Output: $W - K + 1 + 2P$

Very common:

Set $P = (K - 1) / 2$ to
make output have
same size as input!

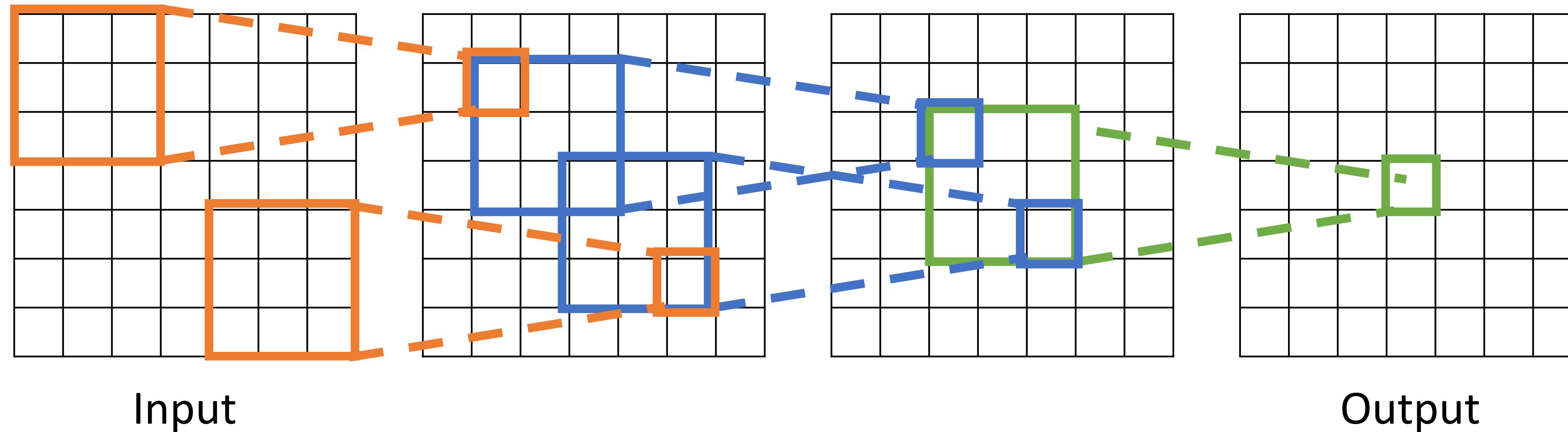
Receptive Fields

For convolution with kernel size K, each element in the output depends on a $K \times K$ **receptive field** in the input



Receptive Fields

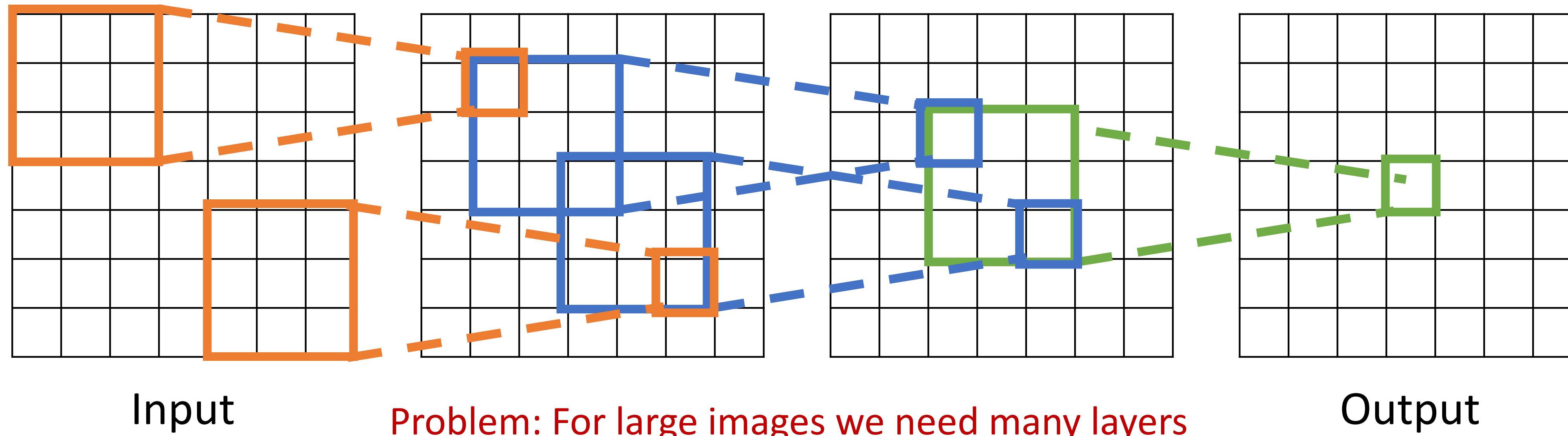
Each successive convolution adds $K - 1$ to the receptive field size
With L layers the receptive field size is $1 + L * (K - 1)$



Be careful – “receptive field in the input” vs “receptive field in the previous layer”
Hopefully clear from context!

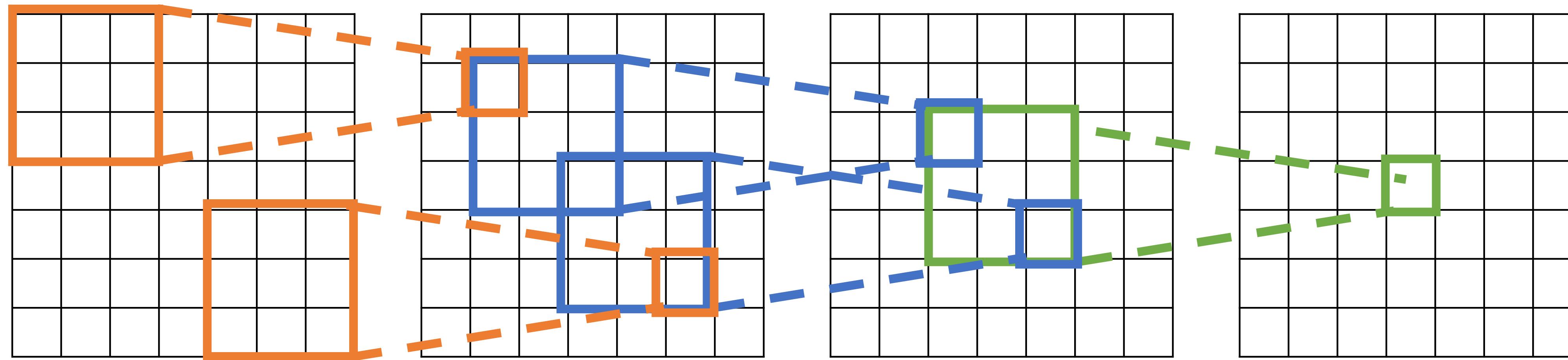
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size
With L layers the receptive field size is $1 + L * (K - 1)$



Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size
With L layers the receptive field size is $1 + L * (K - 1)$



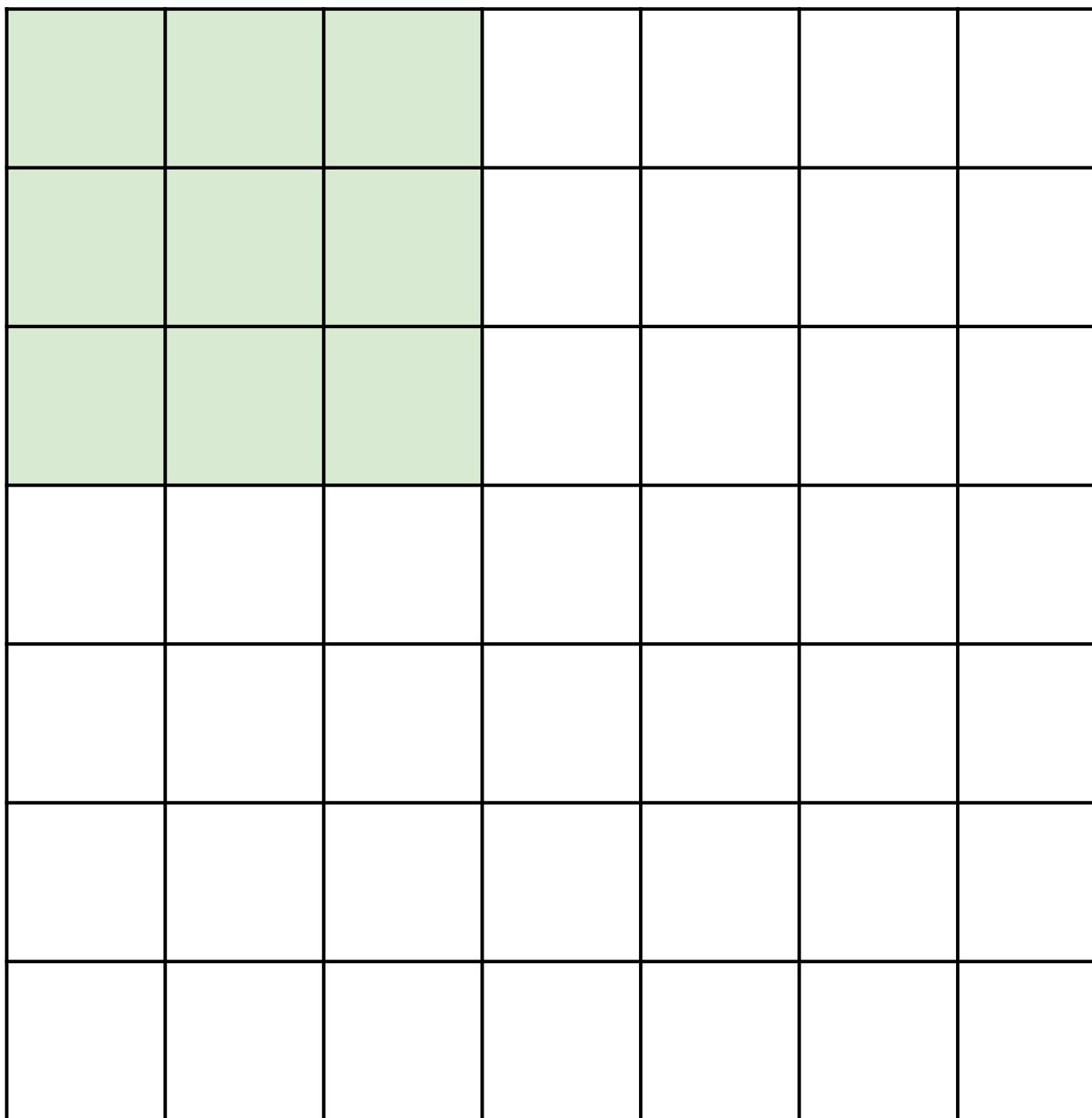
Input

Problem: For large images we need many layers
for each output to “see” the whole image

Output

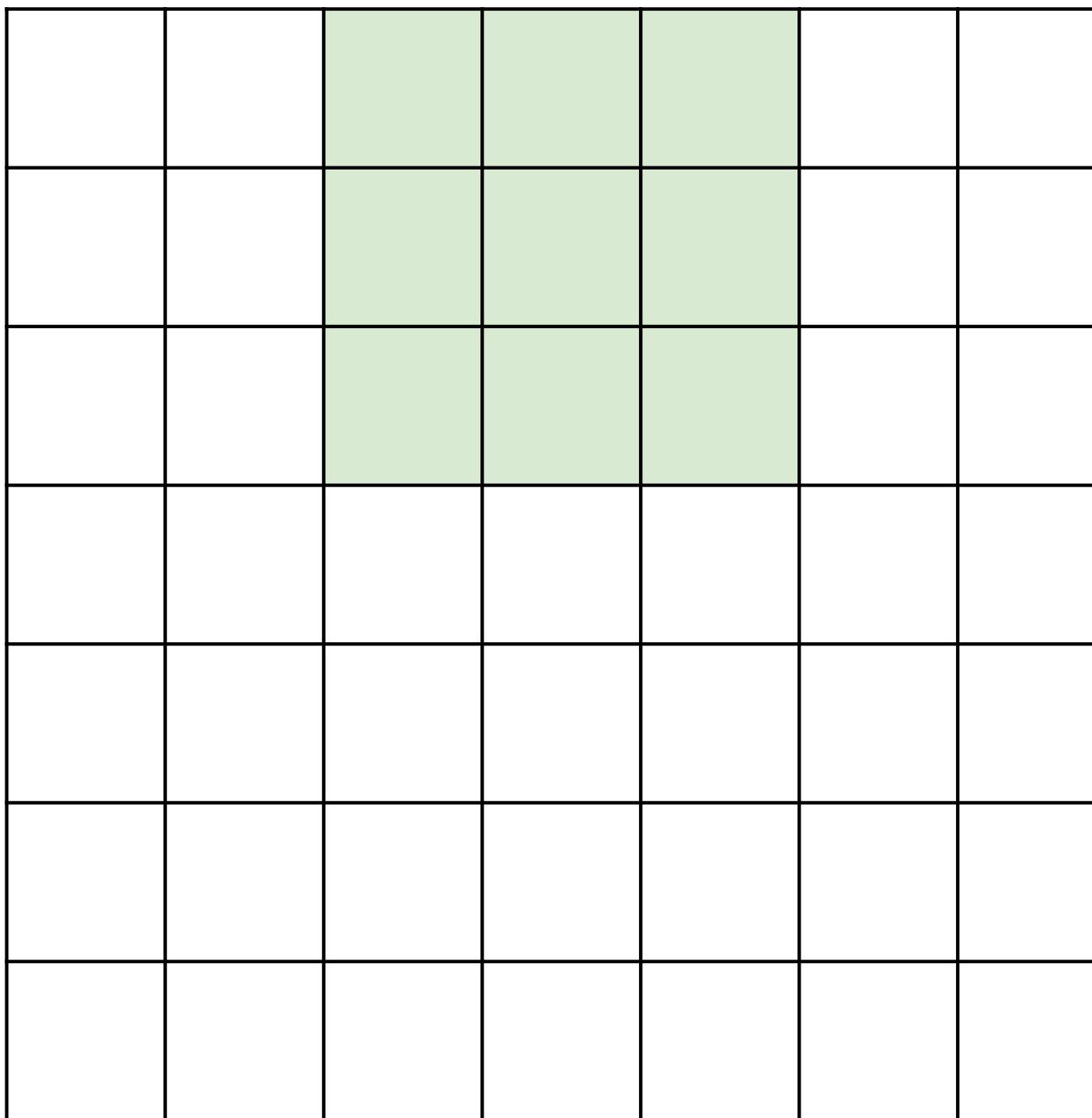
Solution: Downsample inside the network

Strided Convolution



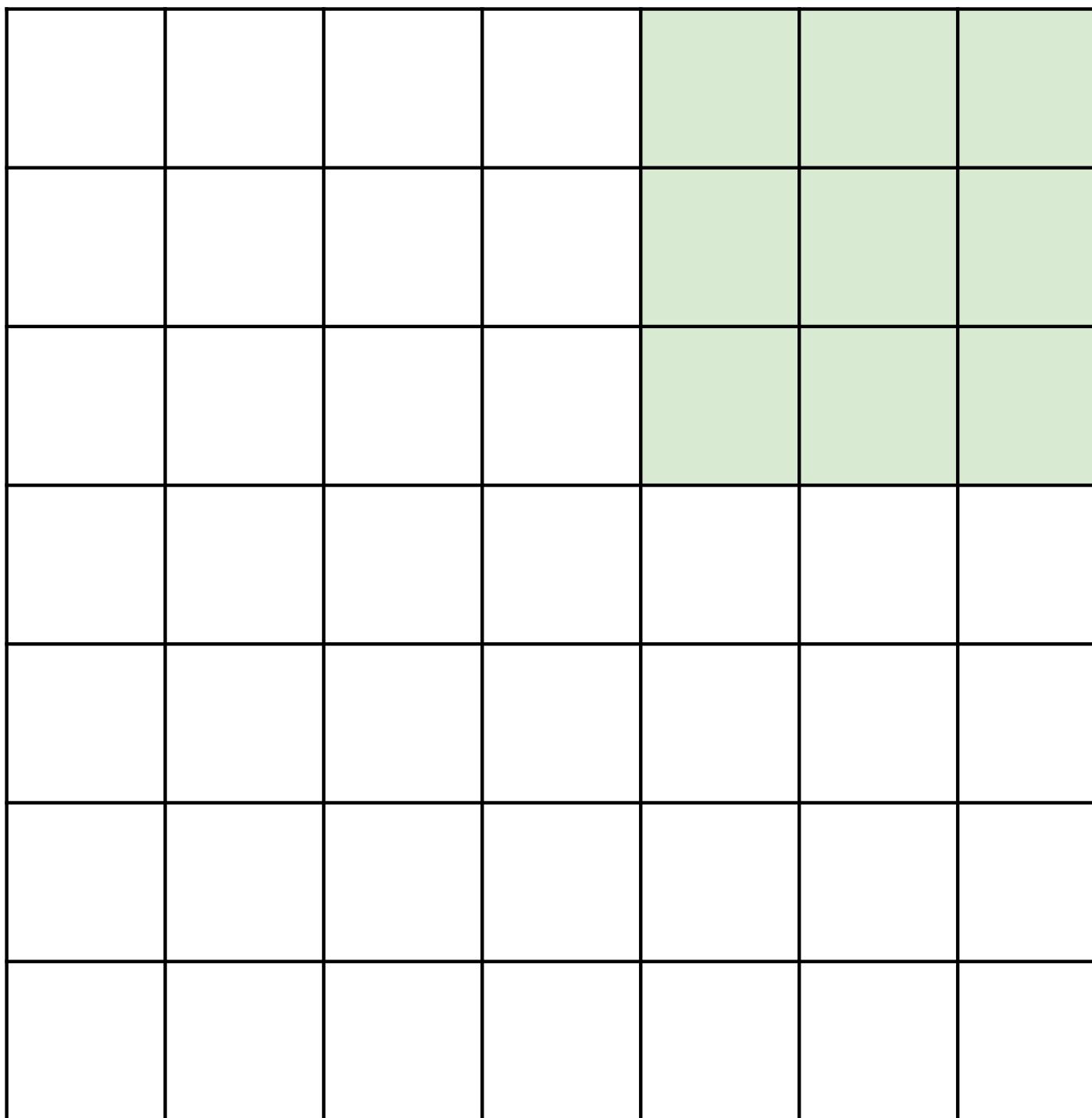
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



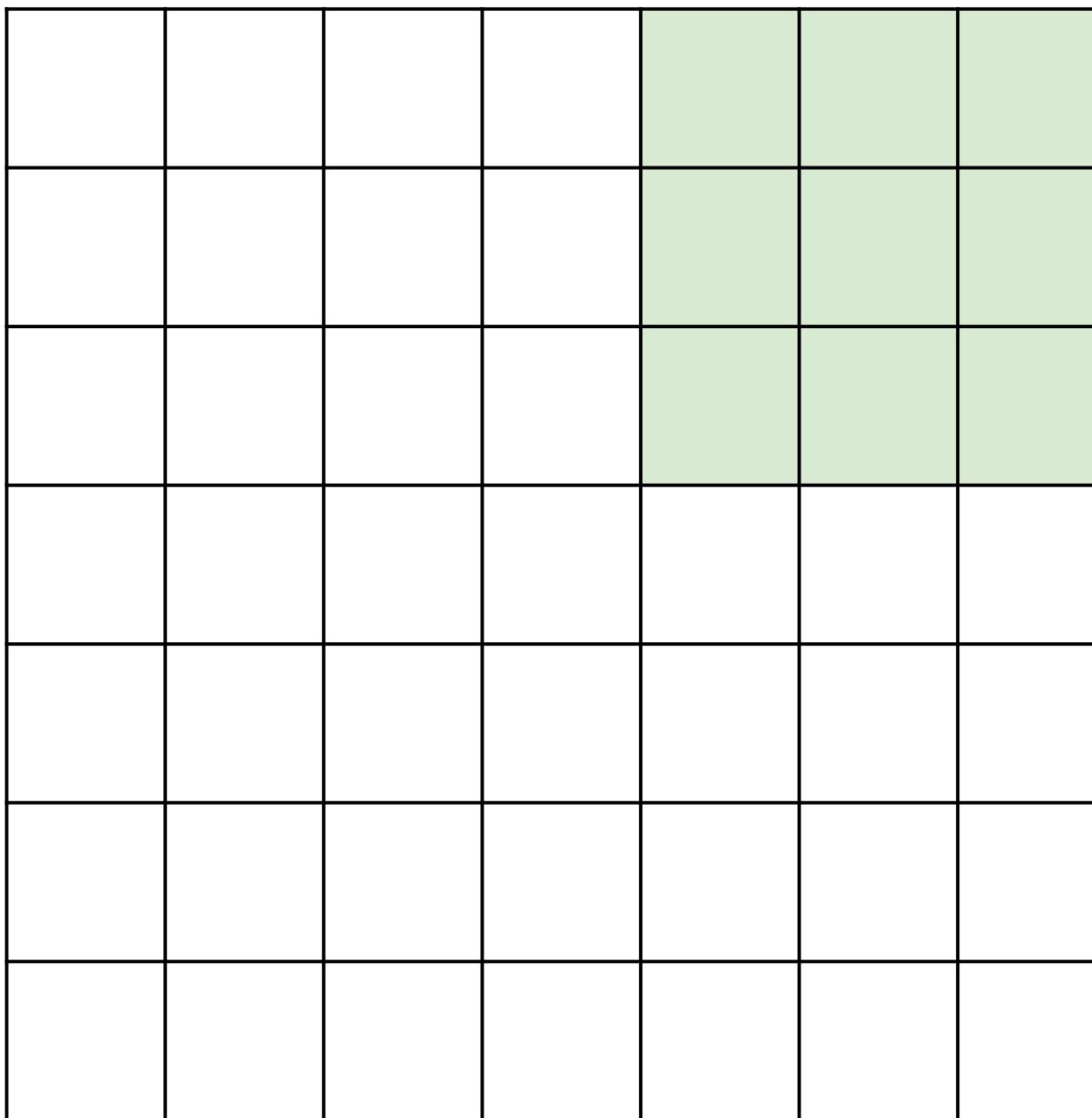
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

In general:

Input: W

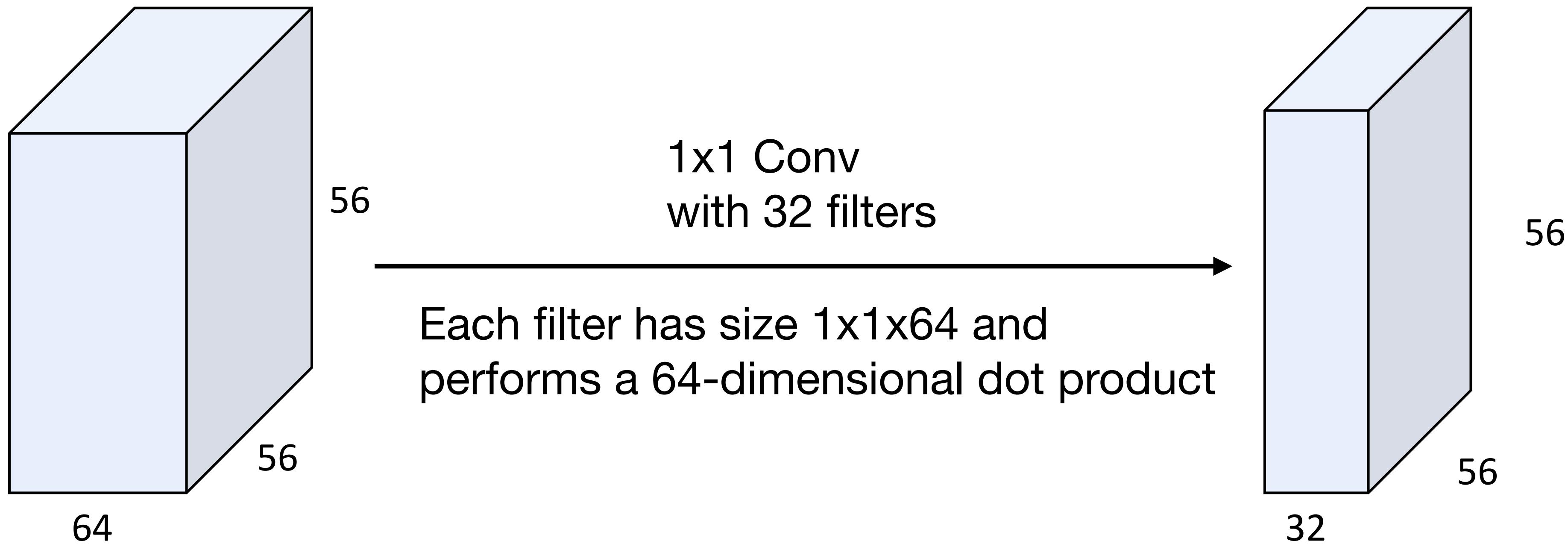
Filter: K

Padding: P

Stride: S

Output: $(W - K + 2P) / S + 1$

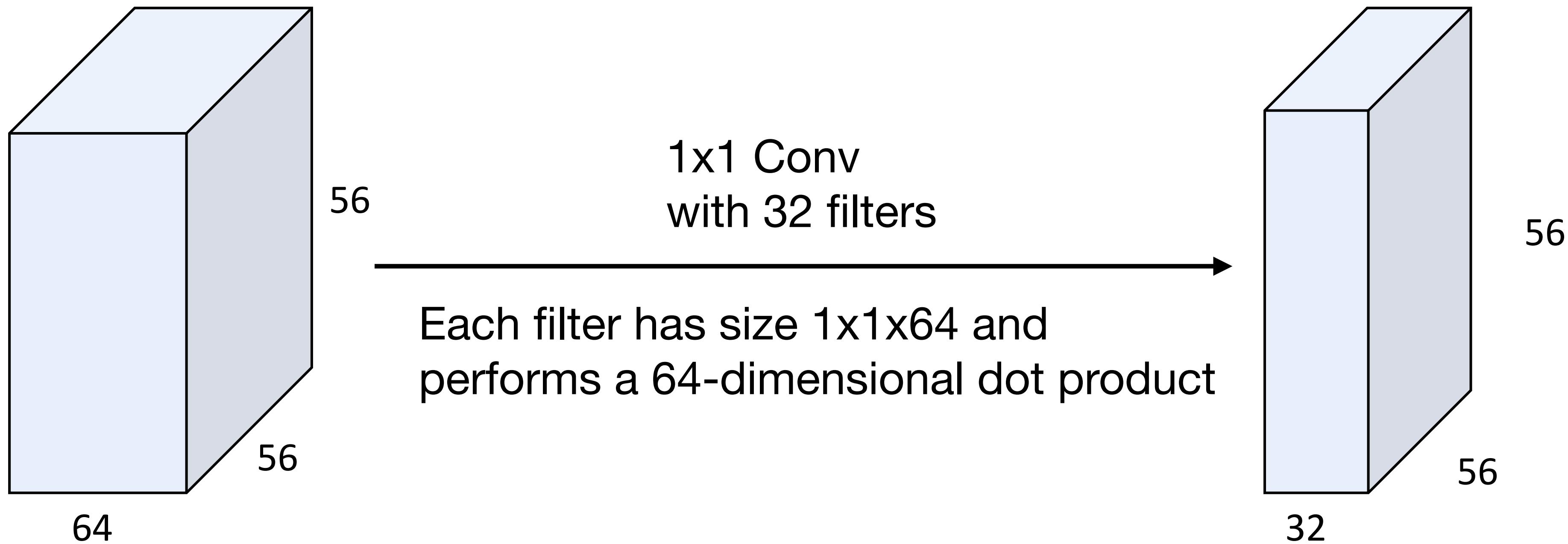
Example: 1x1 Convolution



Stacking 1x1 conv layers gives MLP
operating on each input position



Example: 1x1 Convolution



Convolution Summary

Input: $C_{in} \times H \times W$

Hyperparameters:

- **Kernel size:** $K_H \times K_W$
- **Number filters:** C_{out}
- **Padding:** P
- **Stride:** S

Weight matrix: $C_{out} \times C_{in} \times K_H \times K_W$

giving C_{out} filters of size $C_{in} \times K_H \times K_W$

Bias vector: C_{out}

Output size: $C_{out} \times H' \times W'$ where:

- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$



Convolution Summary

Input: $C_{in} \times H \times W$

Hyperparameters:

- **Kernel size:** $K_H \times K_W$
- **Number filters:** C_{out}
- **Padding:** P
- **Stride:** S

Weight matrix: $C_{out} \times C_{in} \times K_H \times K_W$
giving C_{out} filters of size $C_{in} \times K_H \times K_W$

Bias vector: C_{out}

Output size: $C_{out} \times H' \times W'$ where:

- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

Common settings:

$K_H = K_W$ (Small square filters)

$P = (K - 1) / 2$ ("Same" padding)

$C_{in}, C_{out} = 32, 64, 128, 256$ (powers of 2)

$K = 3, P = 1, S = 1$ (3x3 conv)

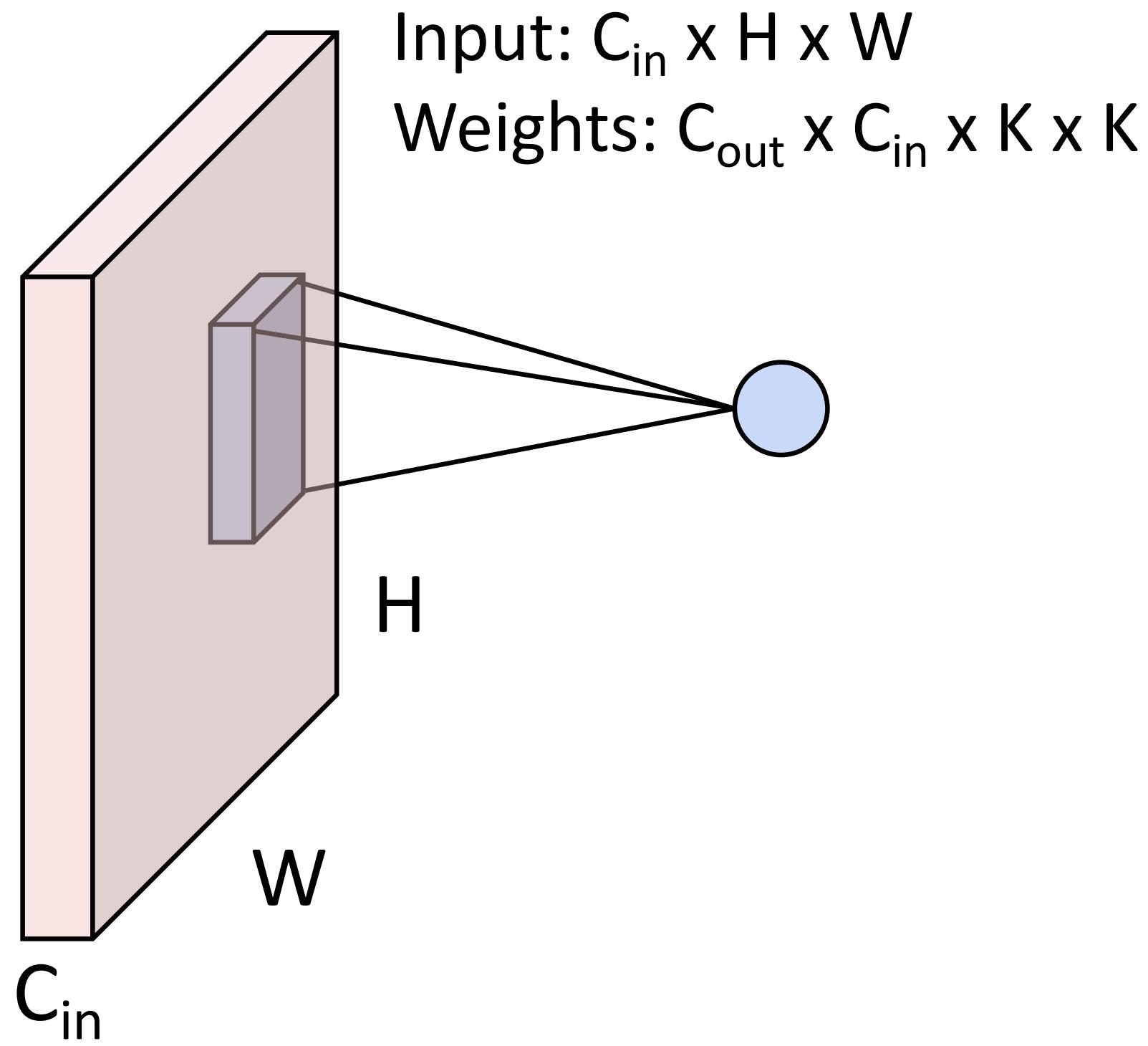
$K = 5, P = 2, S = 1$ (5x5 conv)

$K = 1, P = 0, S = 1$ (1x1 conv)

$K = 3, P = 1, S = 2$ (Downsample by 2)

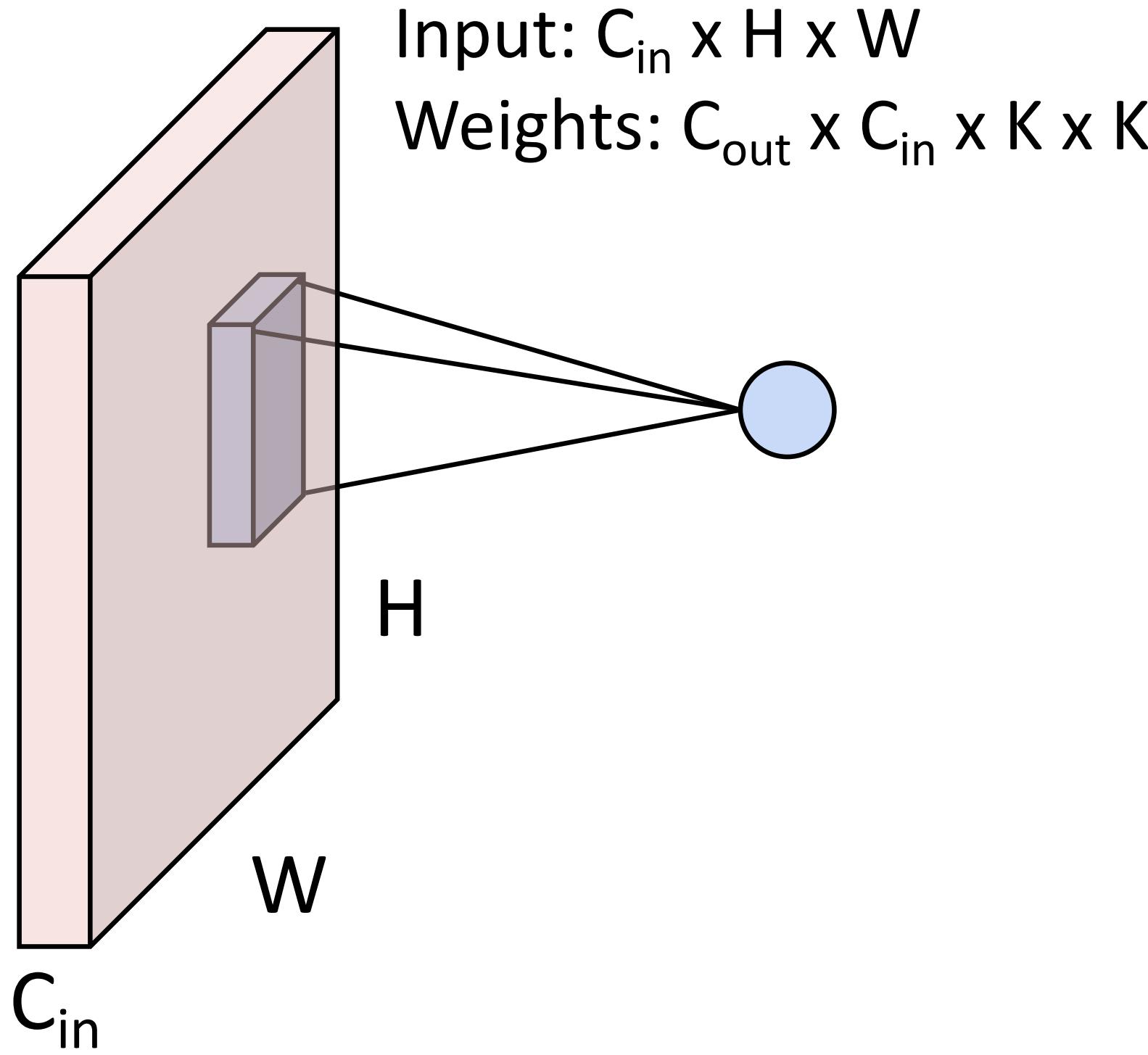
Other types of convolution

So far: 2D Convolution

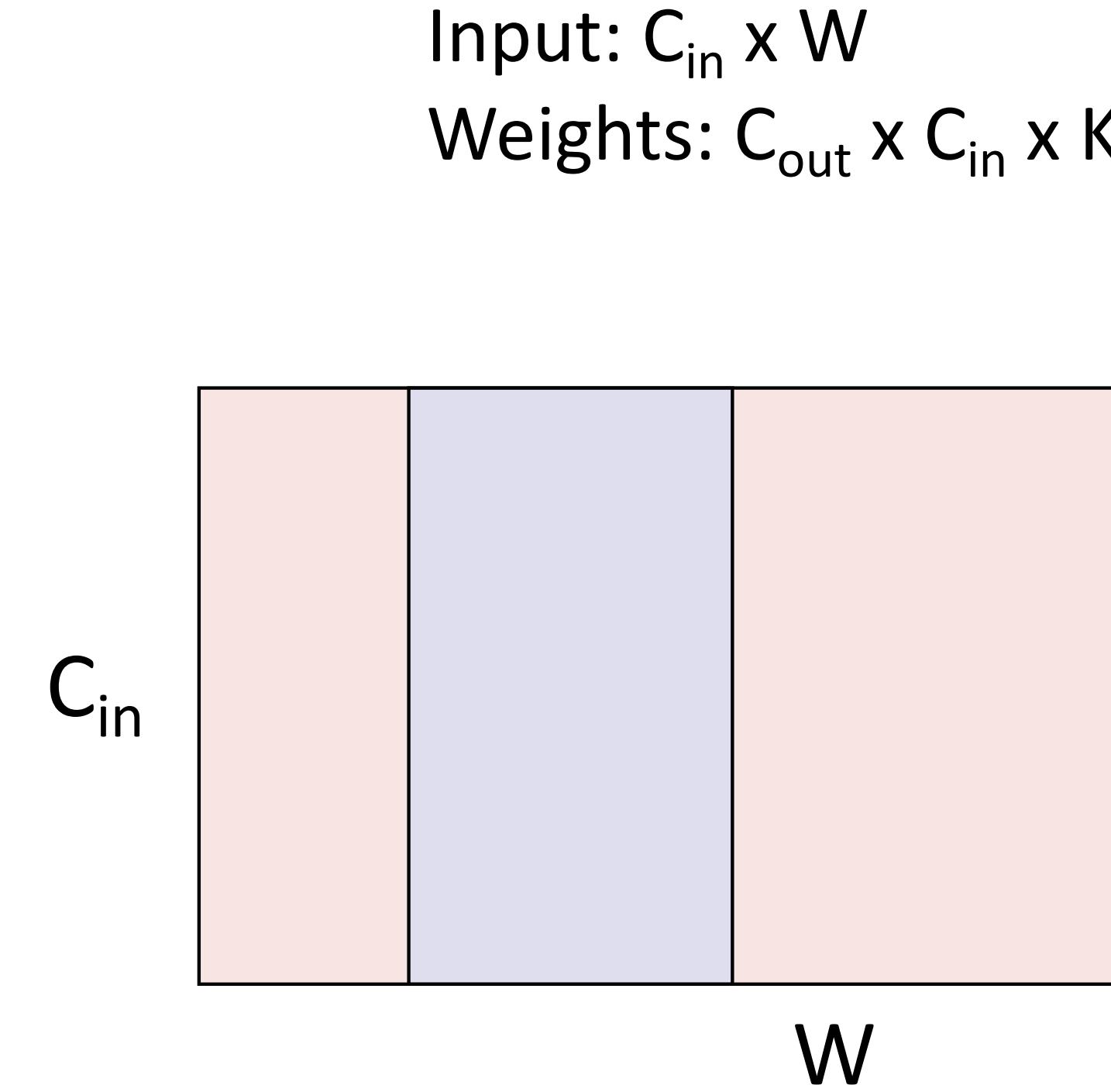


Other types of convolution

So far: 2D Convolution

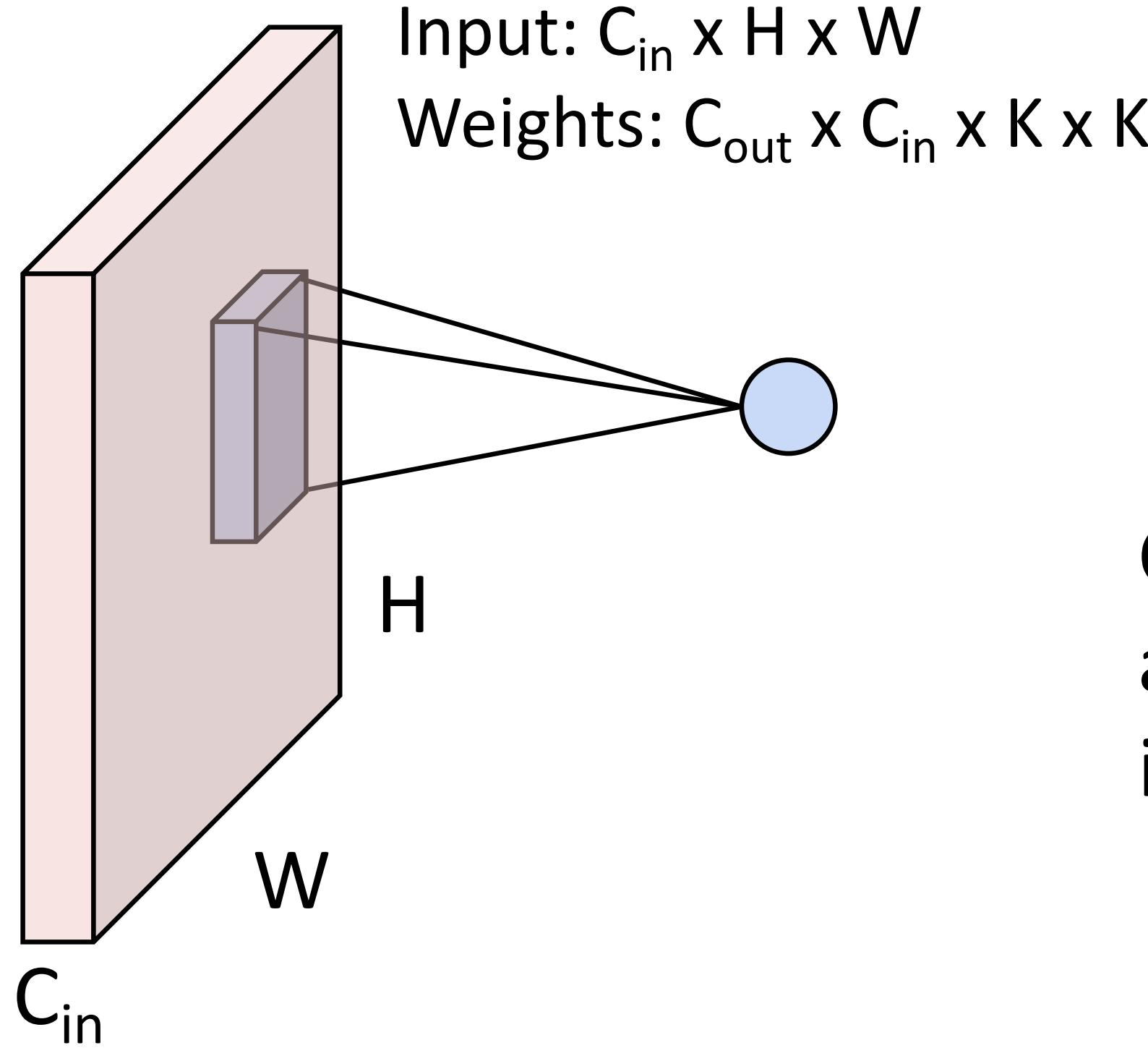


1D Convolution



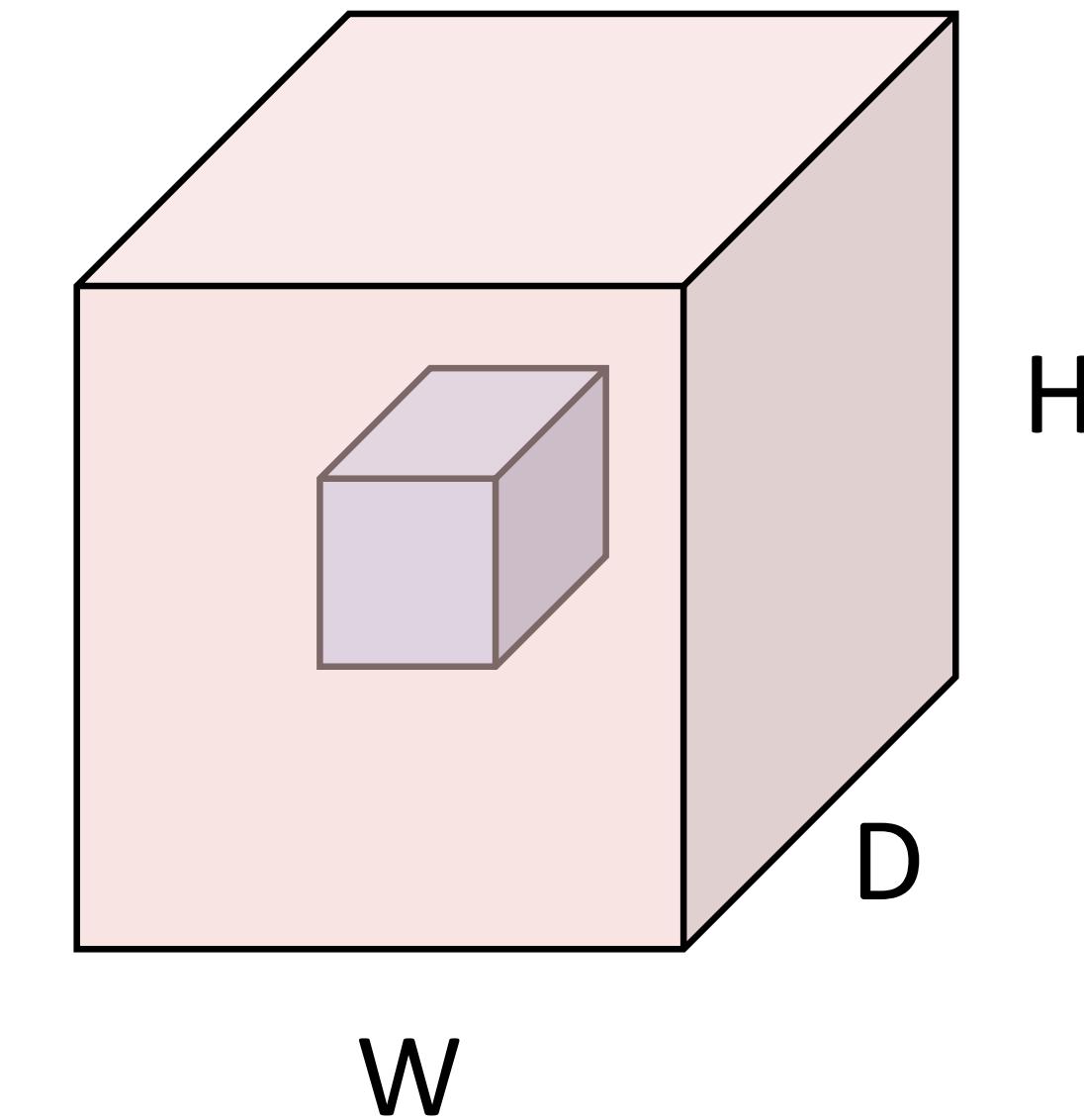
Other types of convolution

So far: 2D Convolution



3D Convolution

Input: $C_{in} \times H \times W \times D$
Weights: $C_{out} \times C_{in} \times K \times K \times K$



C_{in} -dim vector
at each point
in the volume

PyTorch Convolution Layer

Conv2d

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
dilation=1, groups=1, bias=True, padding_mode='zeros')
```

[\[SOURCE\]](#)

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$



PyTorch Convolution Layer

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')`

[SOURCE]

Conv1d

CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')`

[SOURCE] ↗

Conv3d

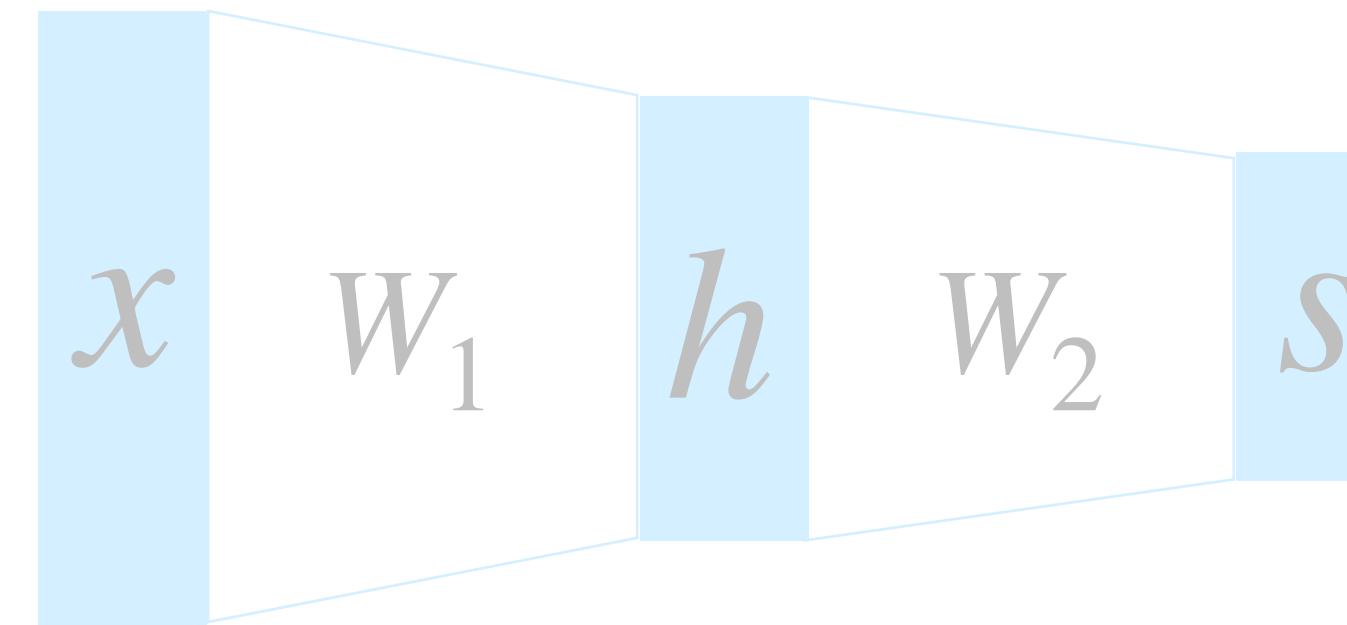
CLASS `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')`

[SOURCE]

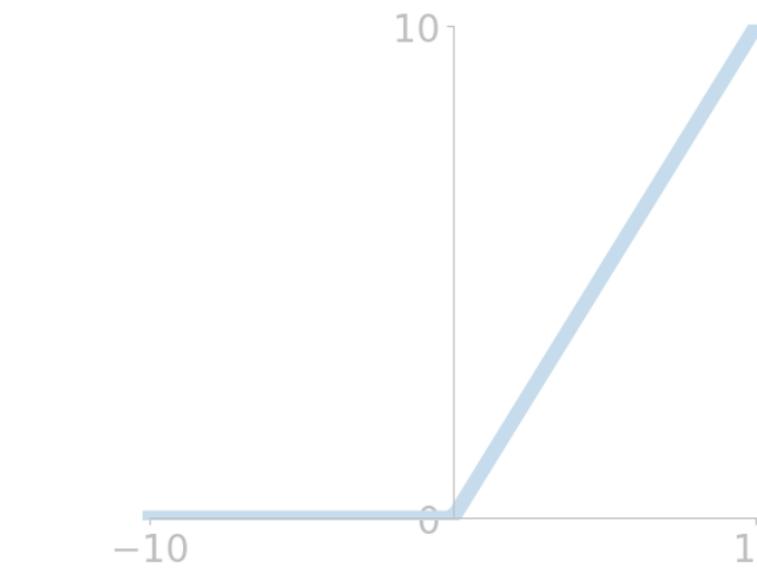


Components of Convolutional Neural Networks

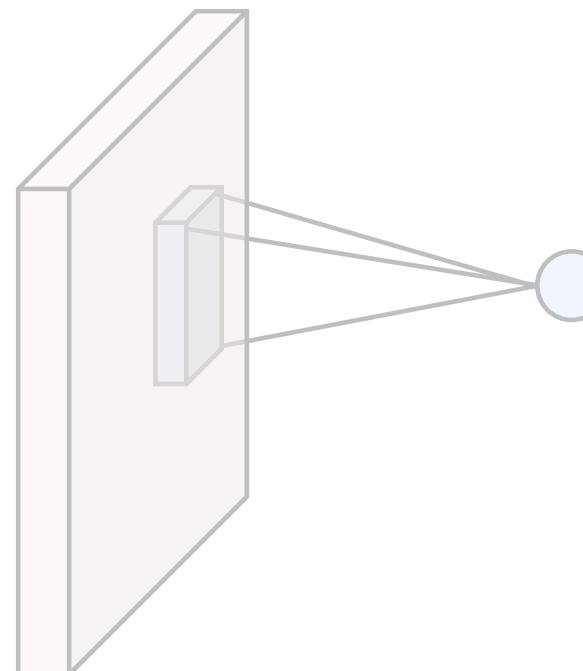
Fully-Connected Layers



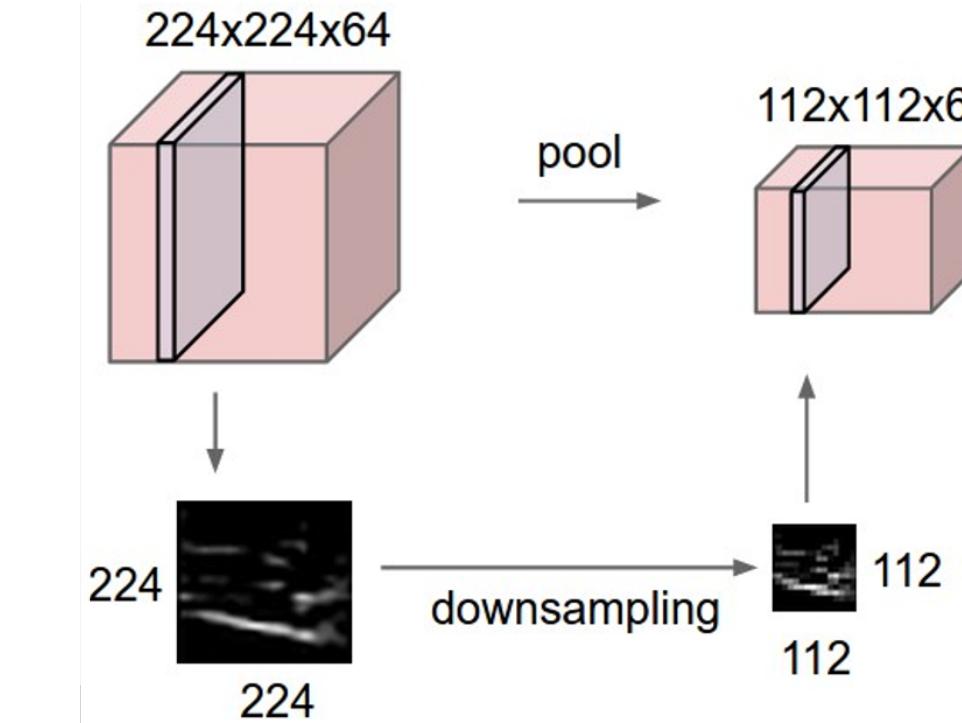
Activation Functions



Convolution Layers



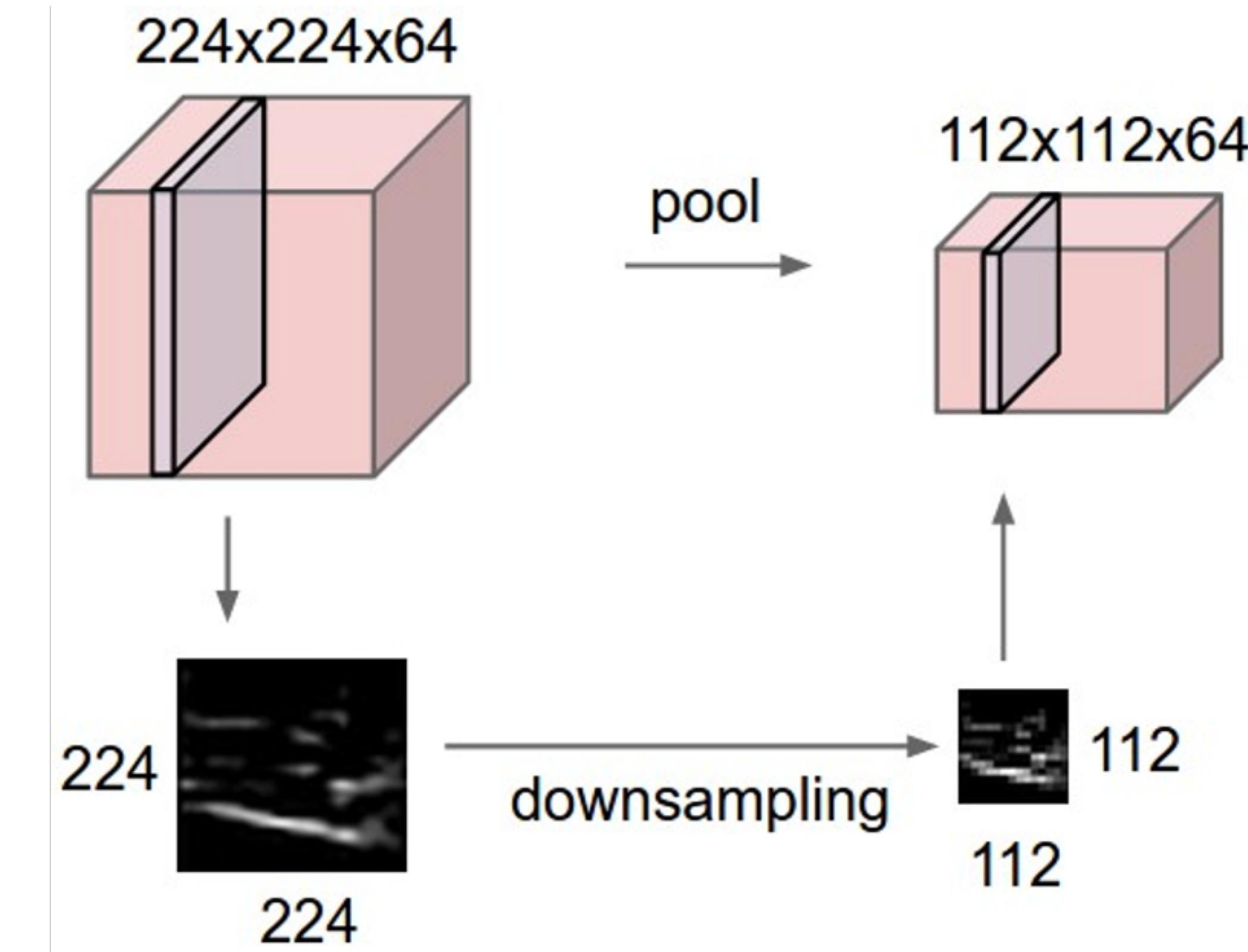
Pooling Layers



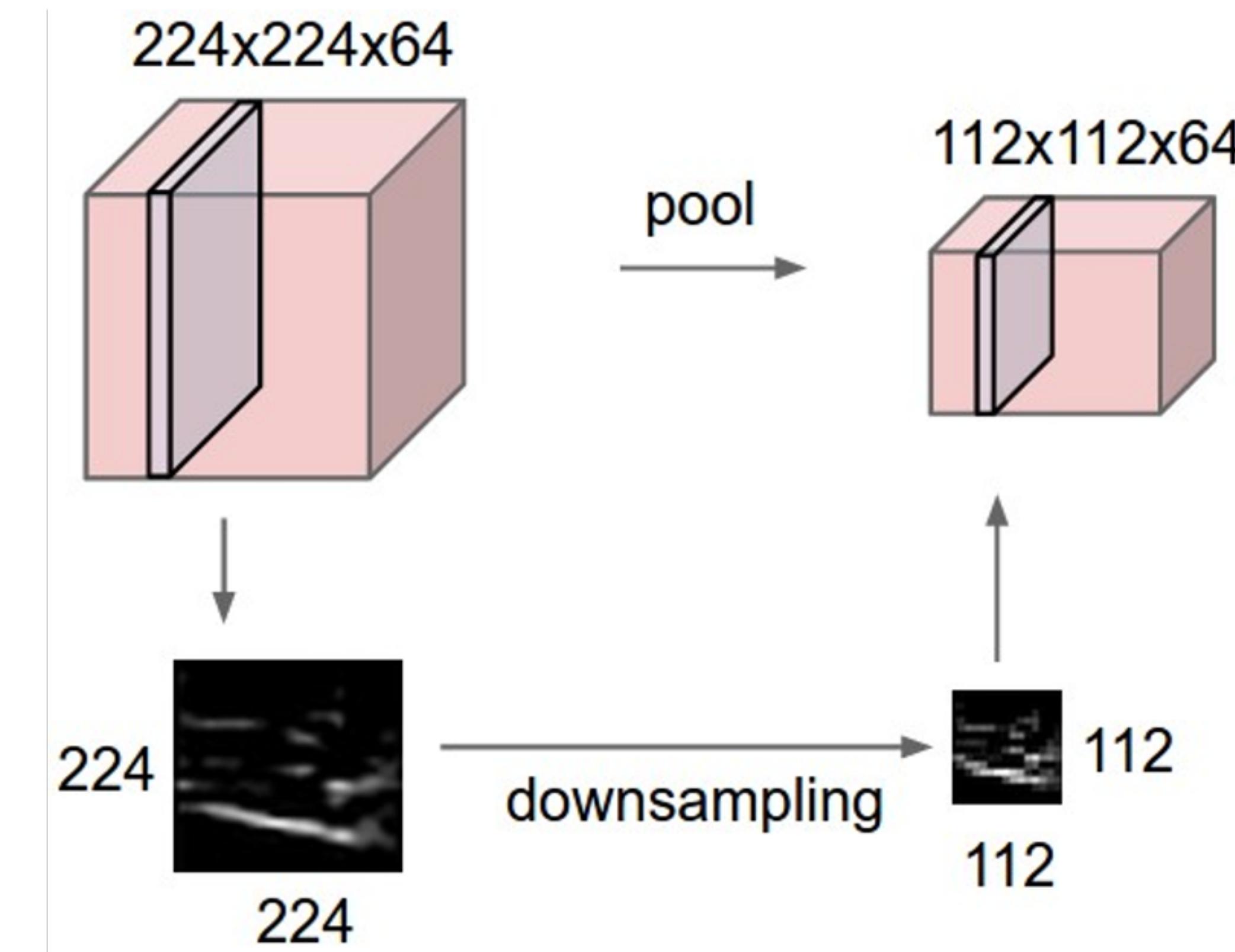
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Pooling Layers: Another way to downsample

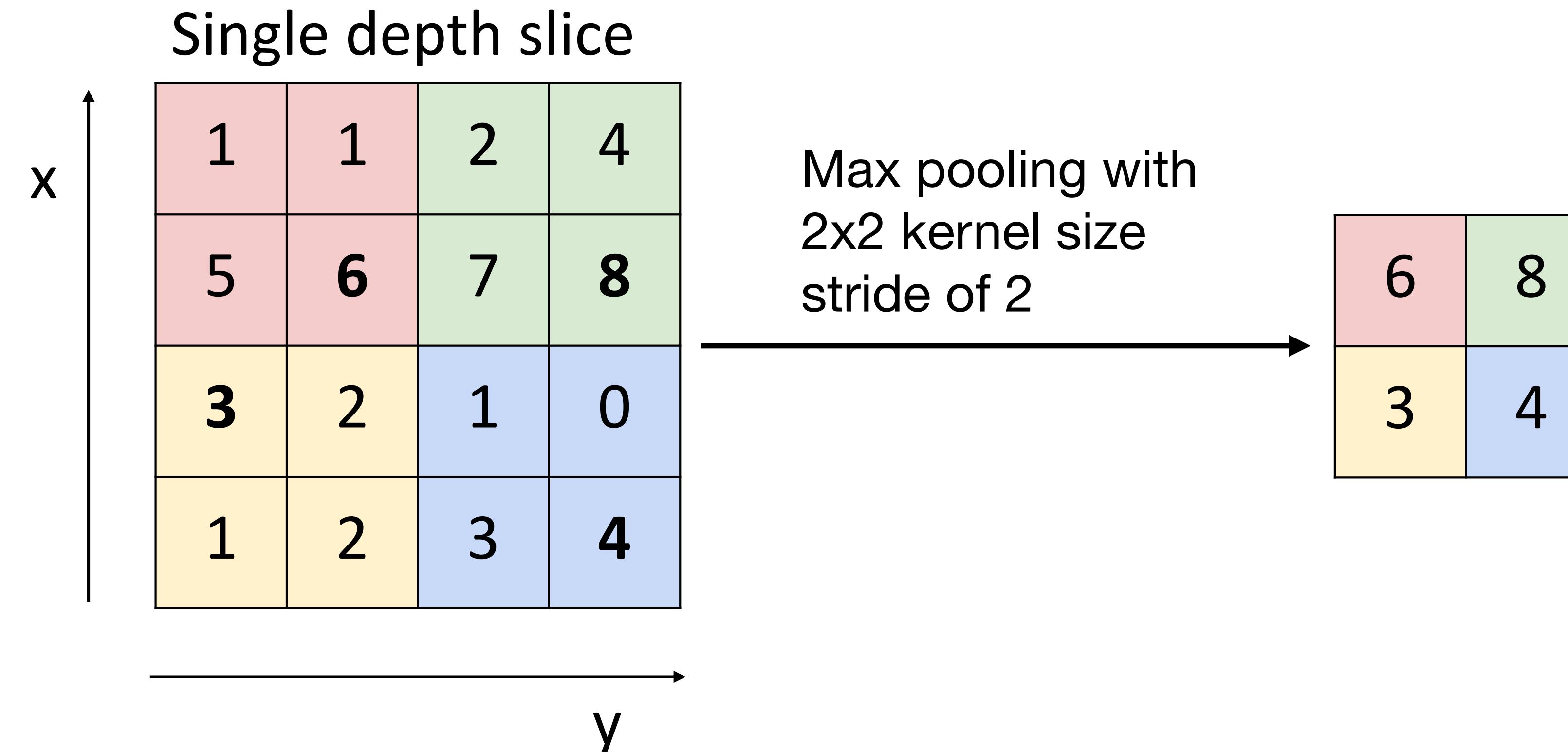


Pooling Layers: Another way to downsample

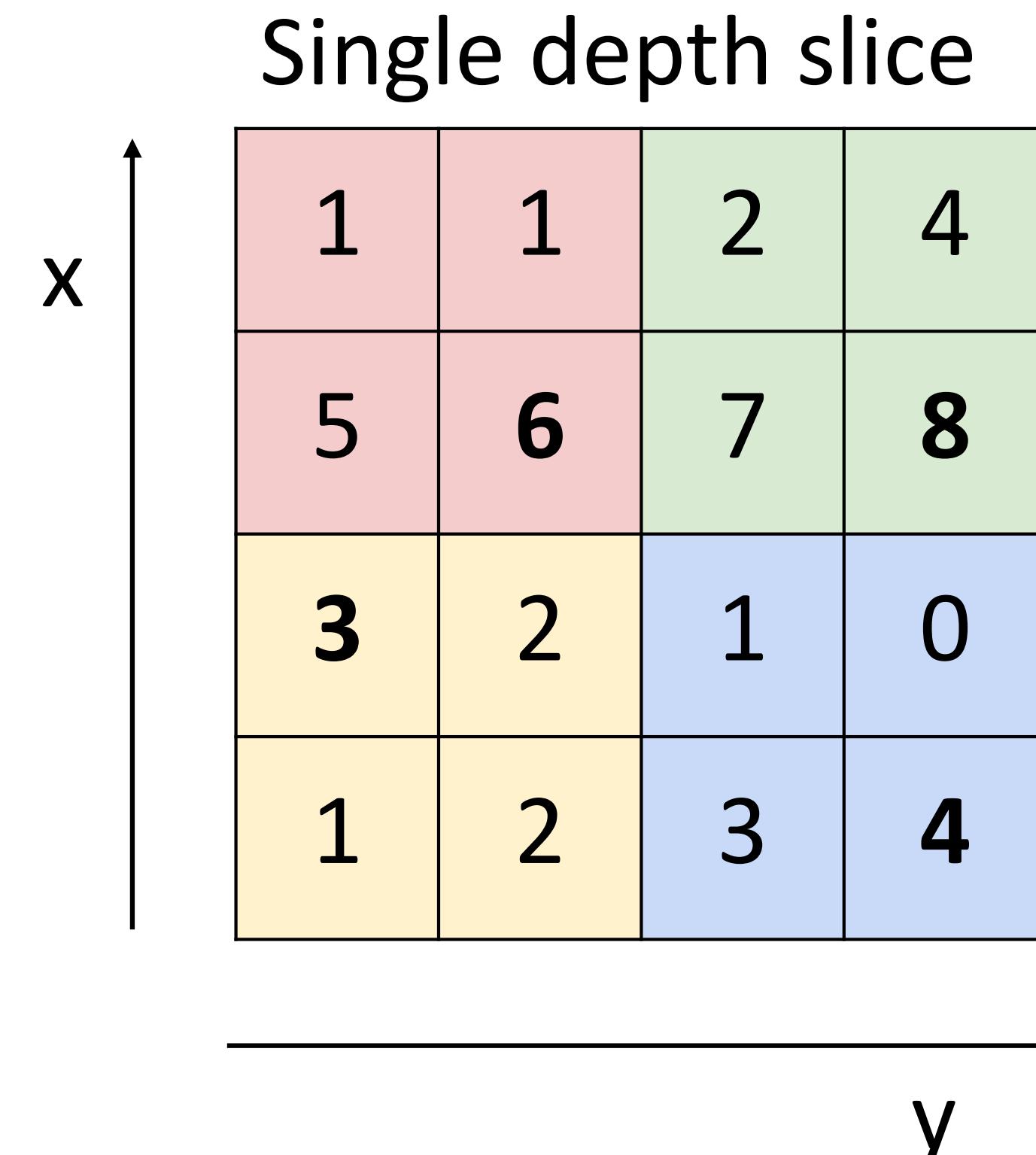


Hyperparameters:
Kernel size
Stride
Pooling function

Max Pooling



Max Pooling



Max pooling with
2x2 kernel size
stride of 2

6	8
3	4

Introduces invariance to
small spatial shifts

No learnable parameters!

Pooling Summary

Input: $C \times H \times W$

Hyperparameters:

- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Common settings:

max, $K = 2, S = 2$

max, $K = 3, S = 2$ (AlexNet)

Output: $C \times H' \times W'$ where

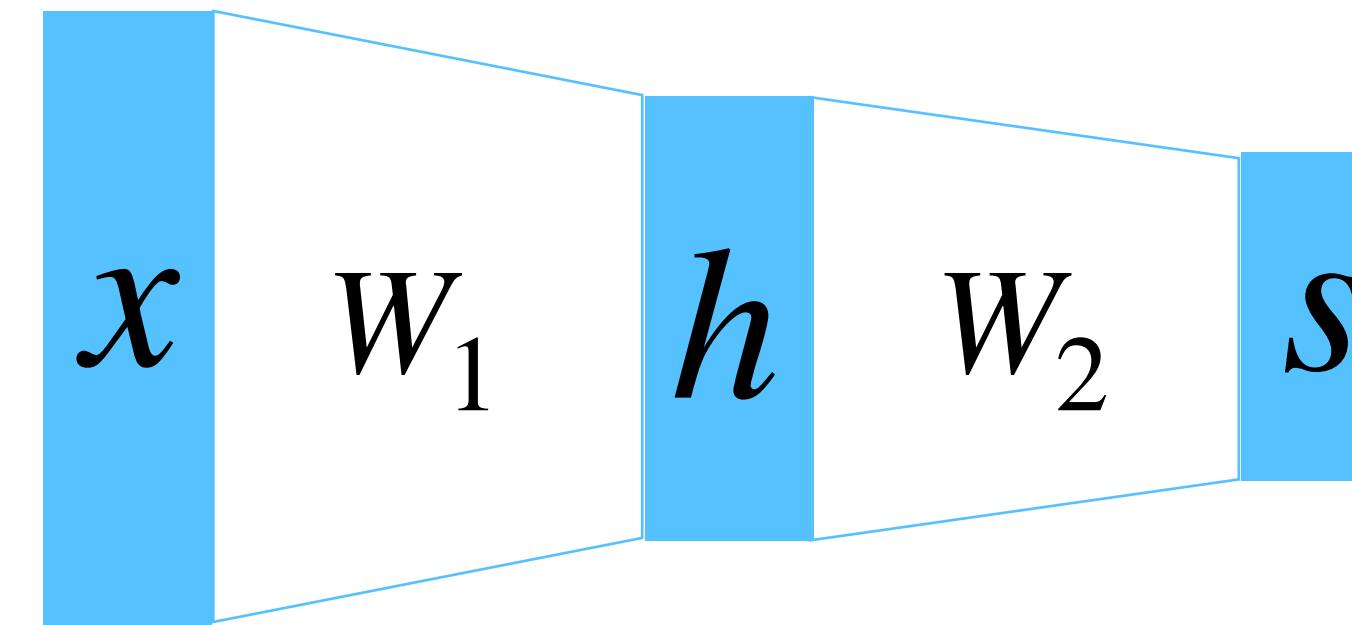
- $H' = (H - K) / S + 1$
- $W' = (W - K) / S + 1$

Learnable parameters: None!

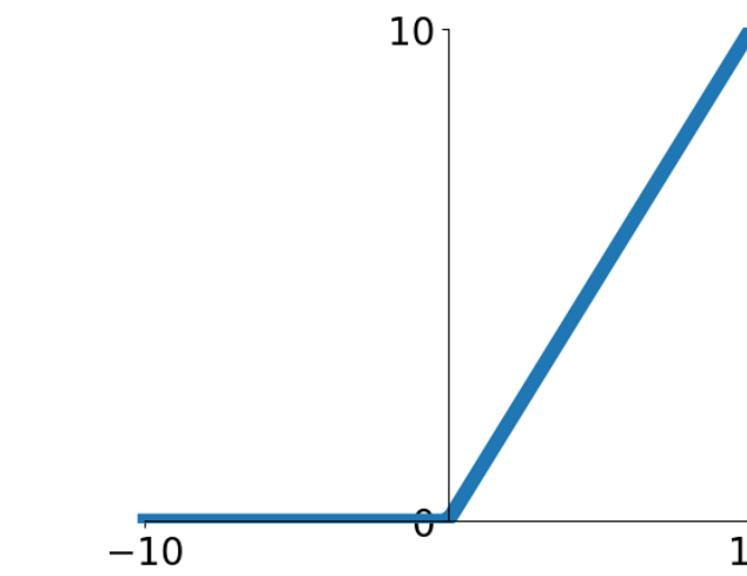


Components of Convolutional Neural Networks

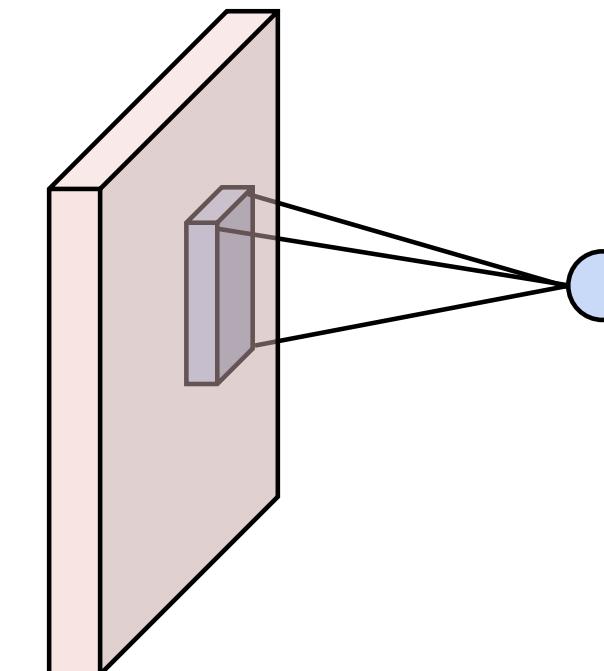
Fully-Connected Layers



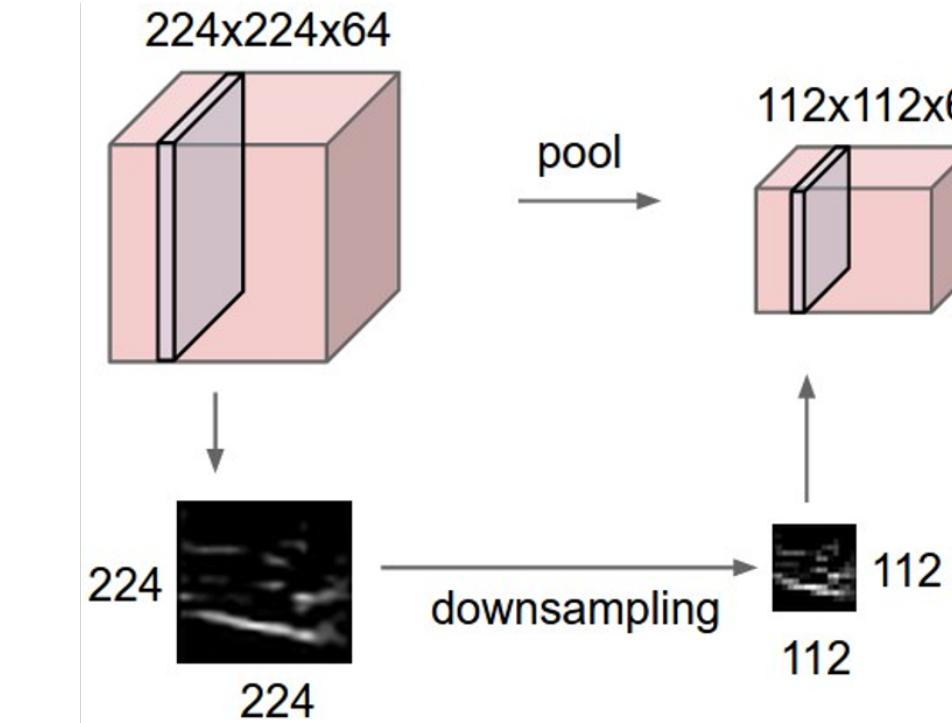
Activation Functions



Convolution Layers



Pooling Layers



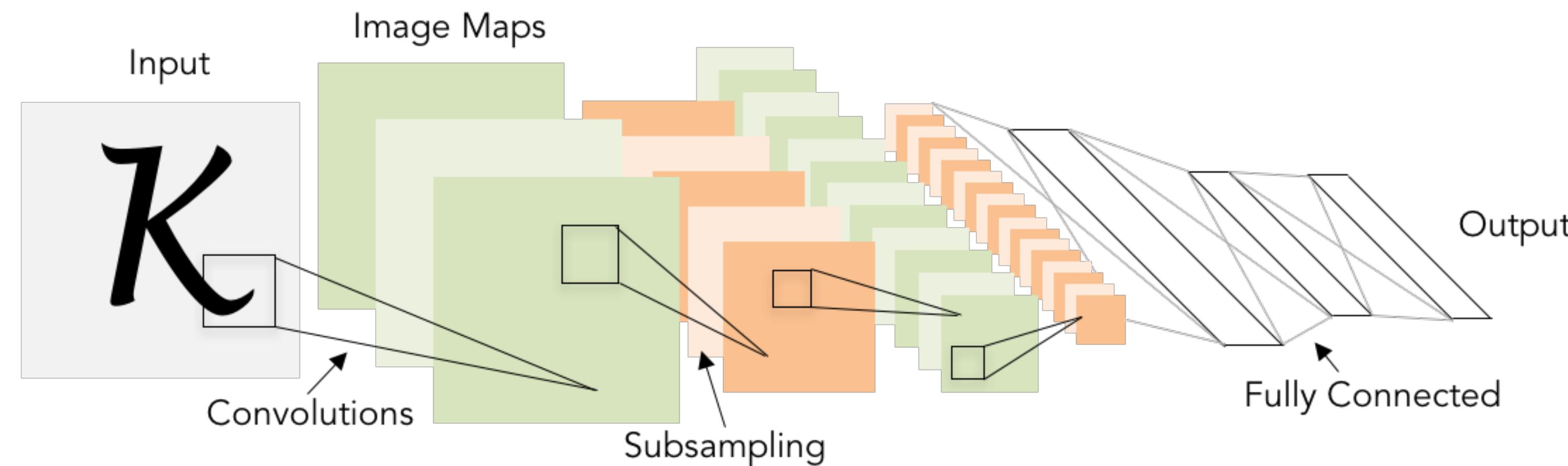
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Convolutional Networks

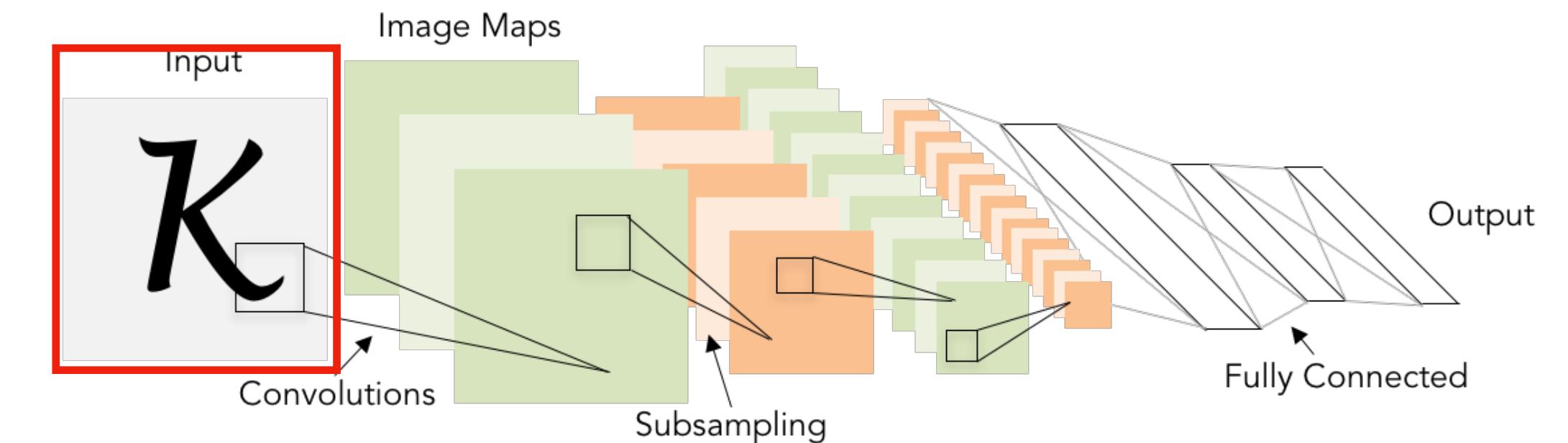
Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

Example: LeNet-5



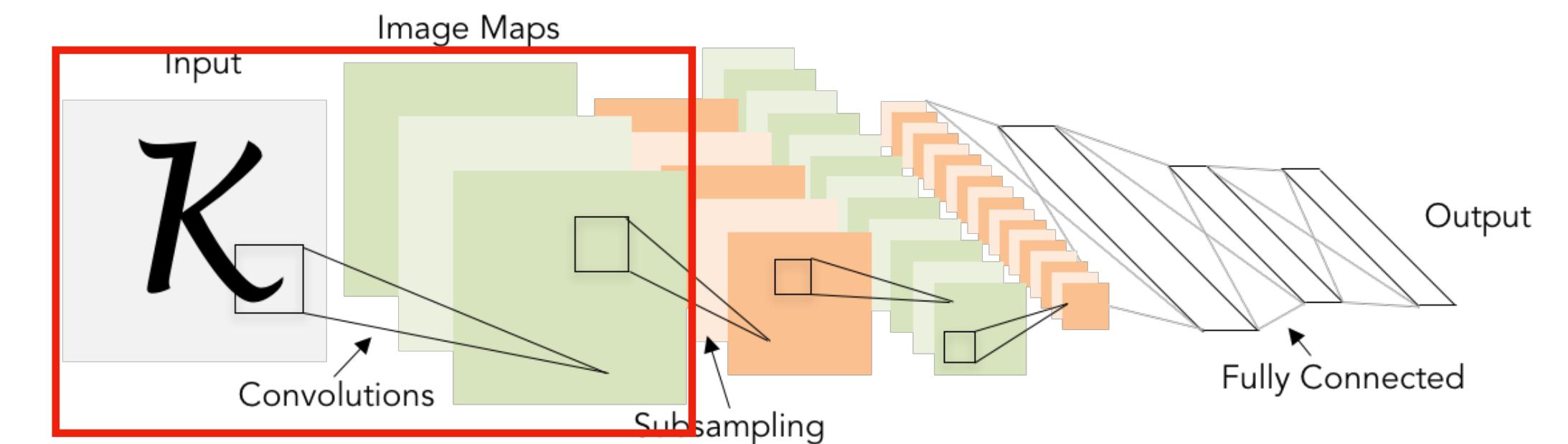
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	



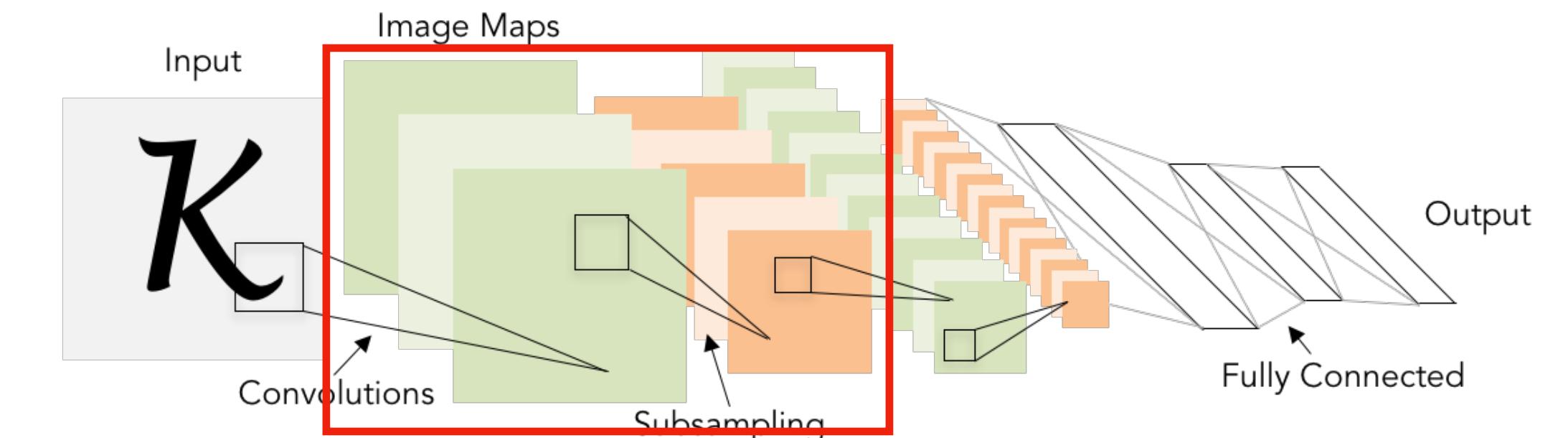
Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	



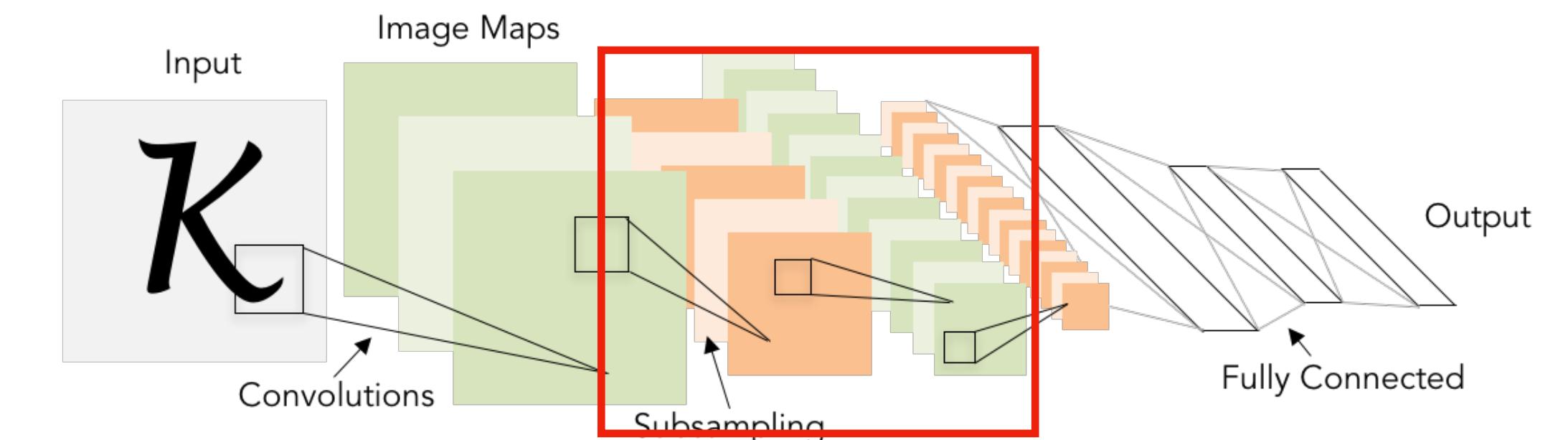
Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	



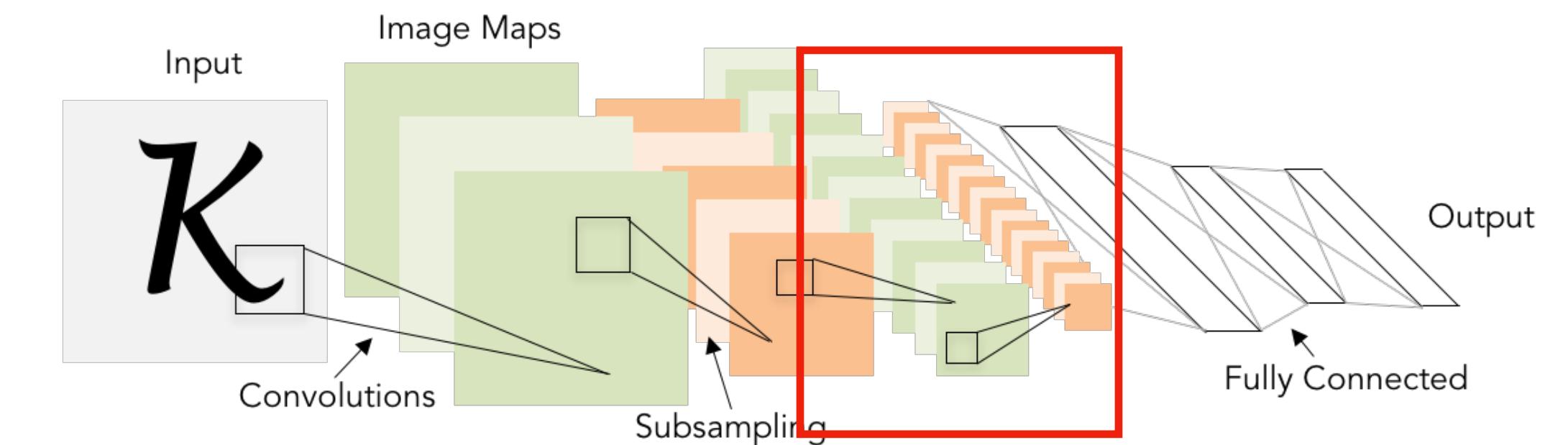
Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	



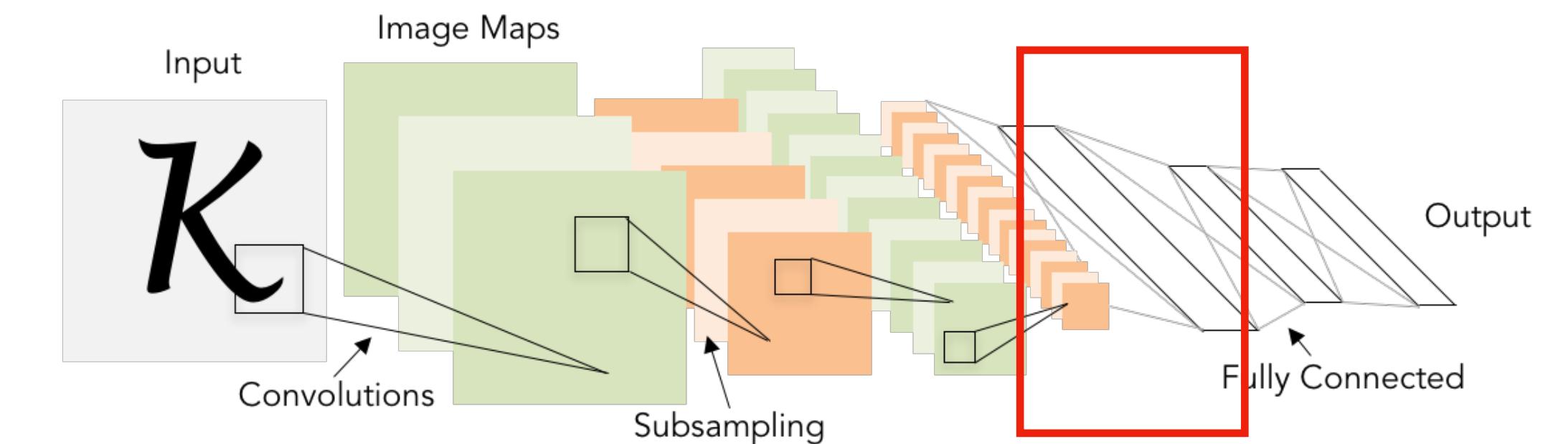
Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	



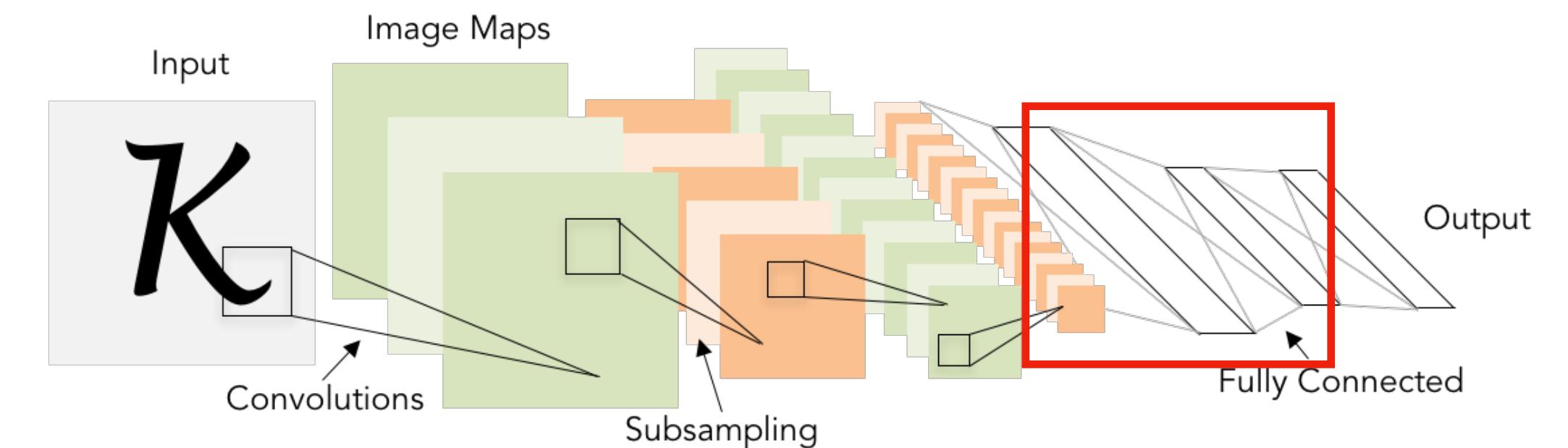
Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	



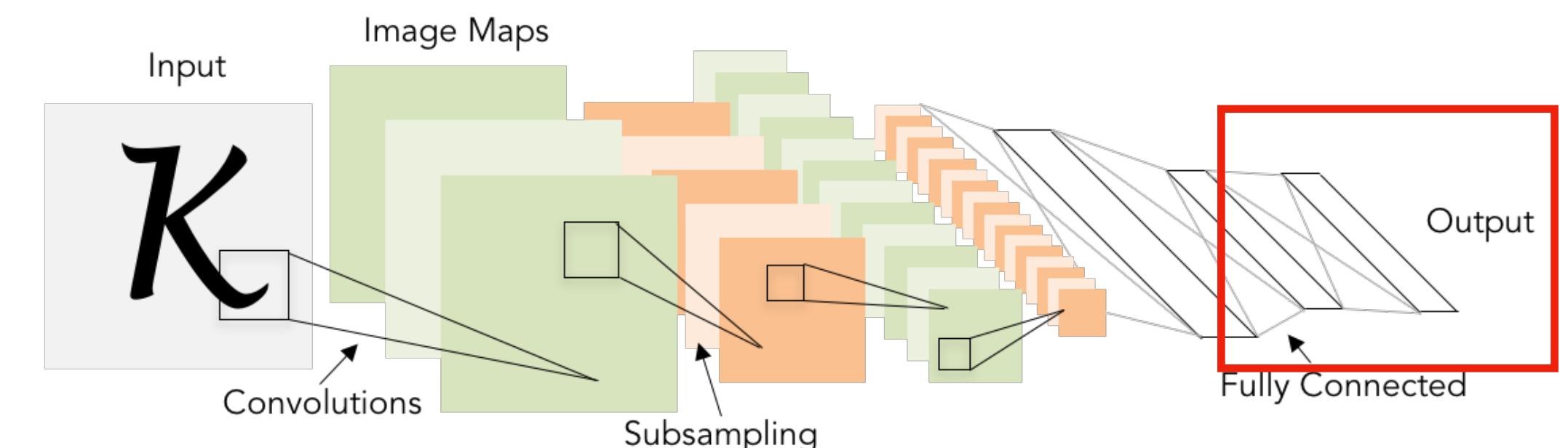
Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear ($2450 \rightarrow 500$)	500	2450×500
ReLU	500	



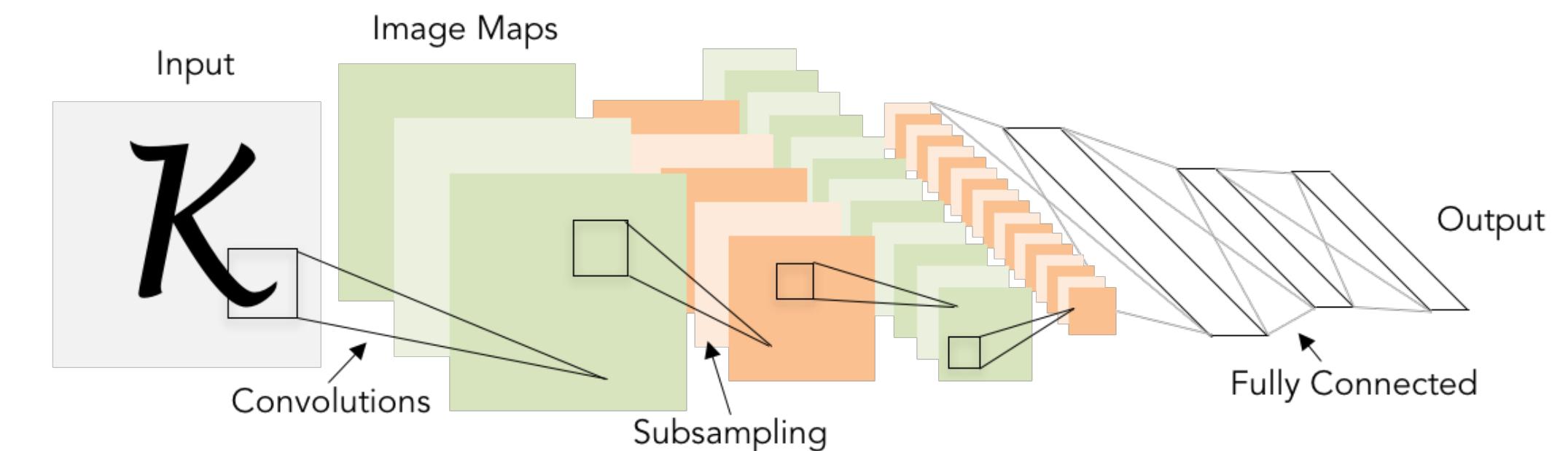
Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear ($2450 \rightarrow 500$)	500	2450×500
ReLU	500	
Linear ($500 \rightarrow 10$)	10	500×10



Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear ($2450 \rightarrow 500$)	500	2450×500
ReLU	500	
Linear ($500 \rightarrow 10$)	10	500×10



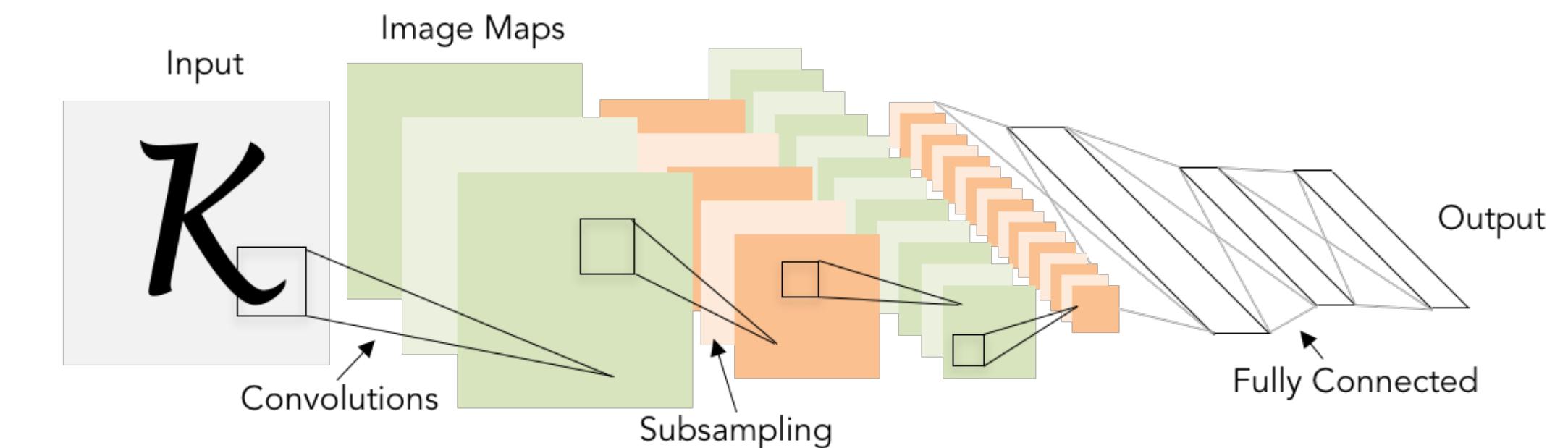
As we progress through the network:

Spatial size decreases
(using pooling or striped convolution)

Number of channels increases
(total “volume” is preserved!)

Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear ($2450 \rightarrow 500$)	500	2450×500
ReLU	500	
Linear ($500 \rightarrow 10$)	10	500×10



As we progress through the network:

Spatial size decreases
(using pooling or striped convolution)

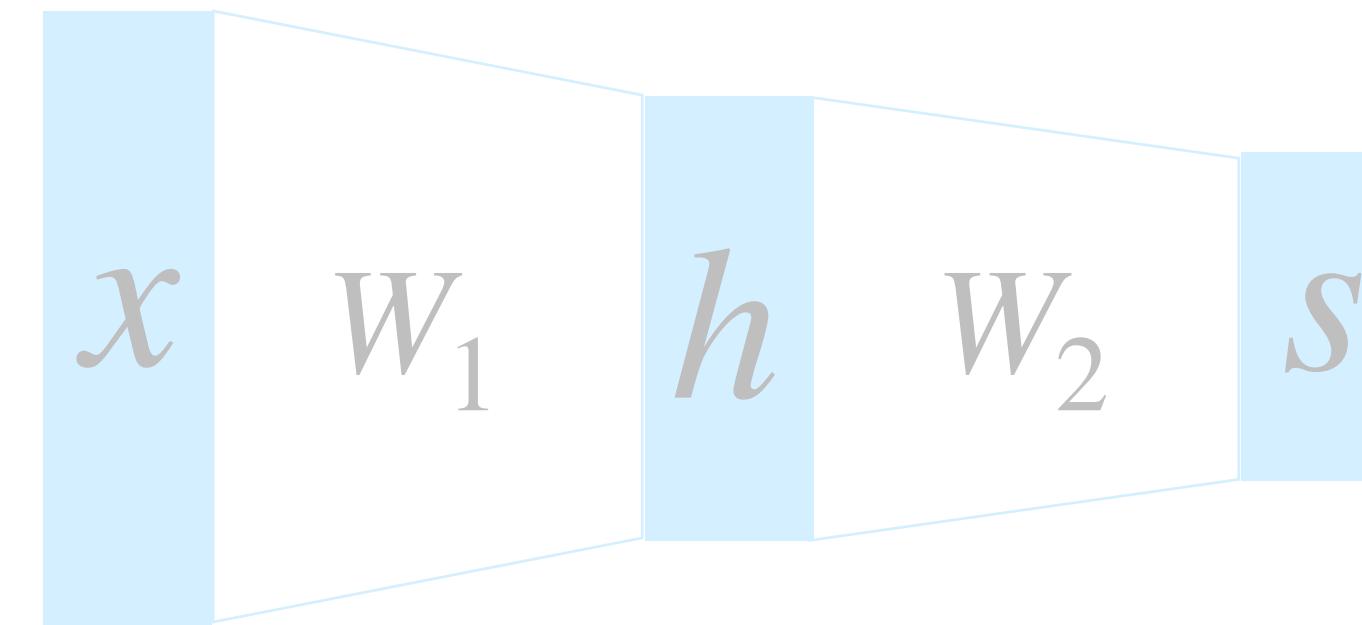
Number of channels increases
(total “volume” is preserved!)

Some modern architectures
break this trend—stay tuned!

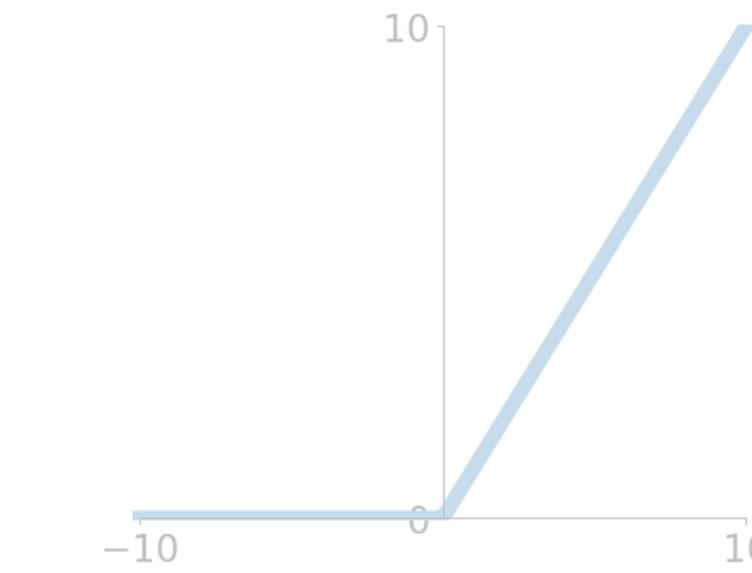
Problem: Deep Networks very hard to train

Components of Convolutional Neural Networks

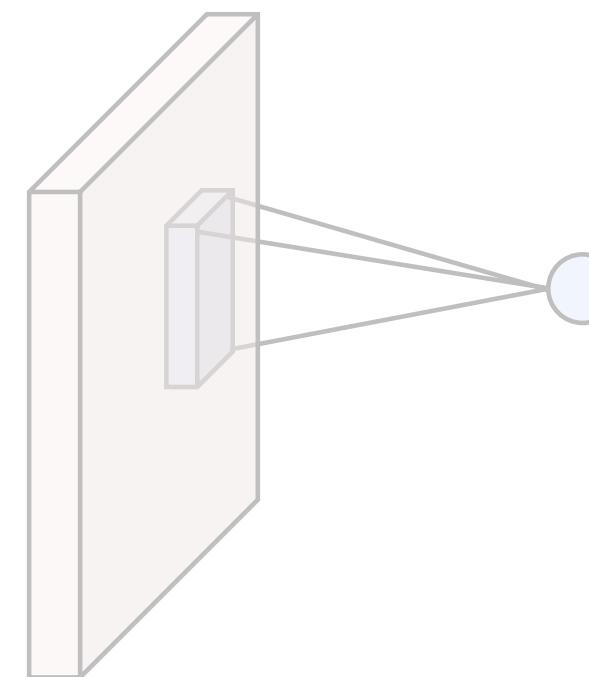
Fully-Connected Layers



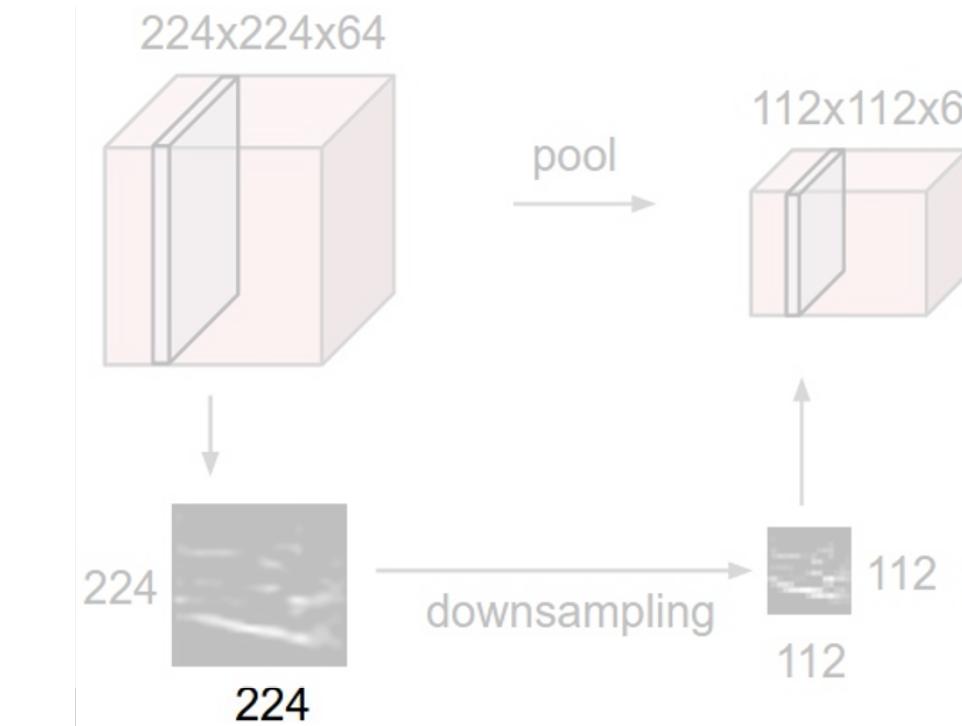
Activation Functions



Convolution Layers



Pooling Layers



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Batch Normalization

Idea: “Normalize” the outputs of a layer so they have zero mean and unit variance

Why? Helps reduce “internal covariate shift”, improves optimization results

We can normalize a batch of activations using:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$



Batch Normalization

Idea: “Normalize” the outputs of a layer so they have zero mean and unit variance

Why? Helps reduce “internal covariate shift”, improves optimization results

We can normalize a batch of activations using:

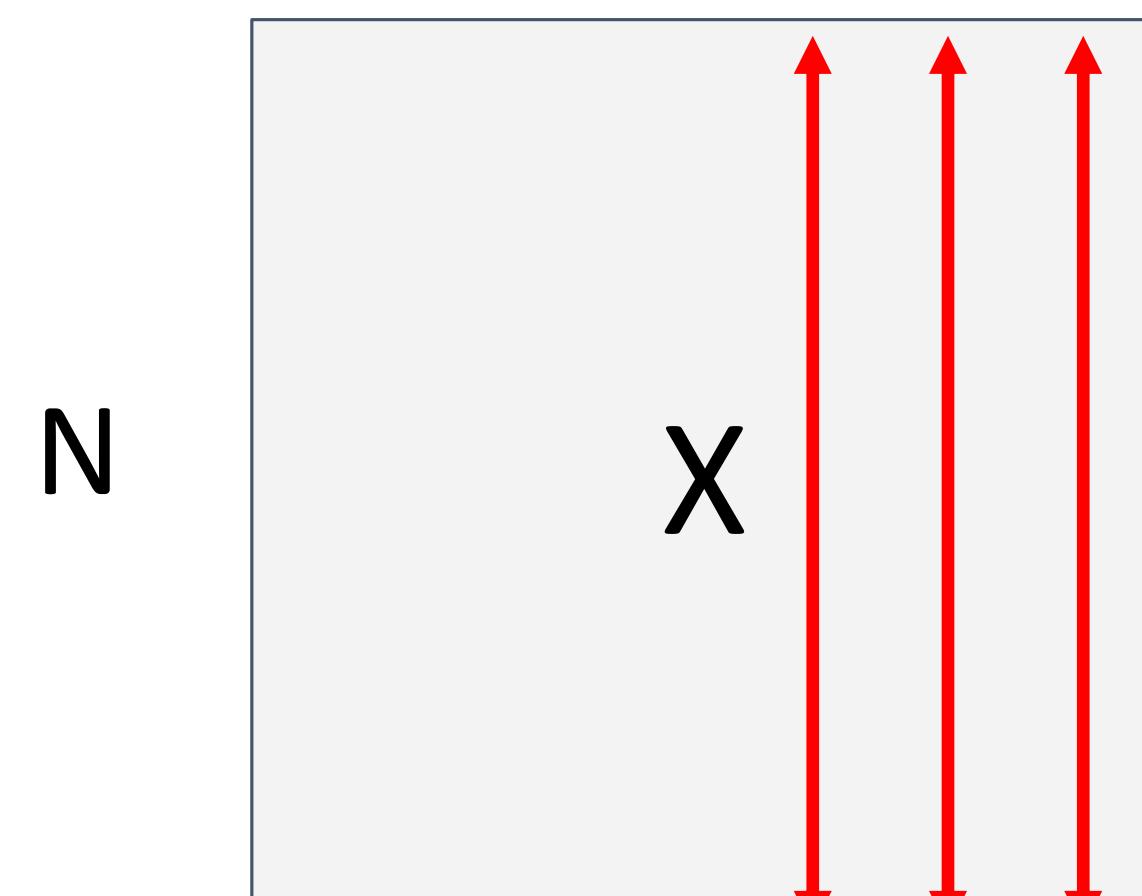
$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

This is a **differentiable function**, so we can use it as an operator in our networks and backprop through it!



Batch Normalization

Input: $x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Per-channel
mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

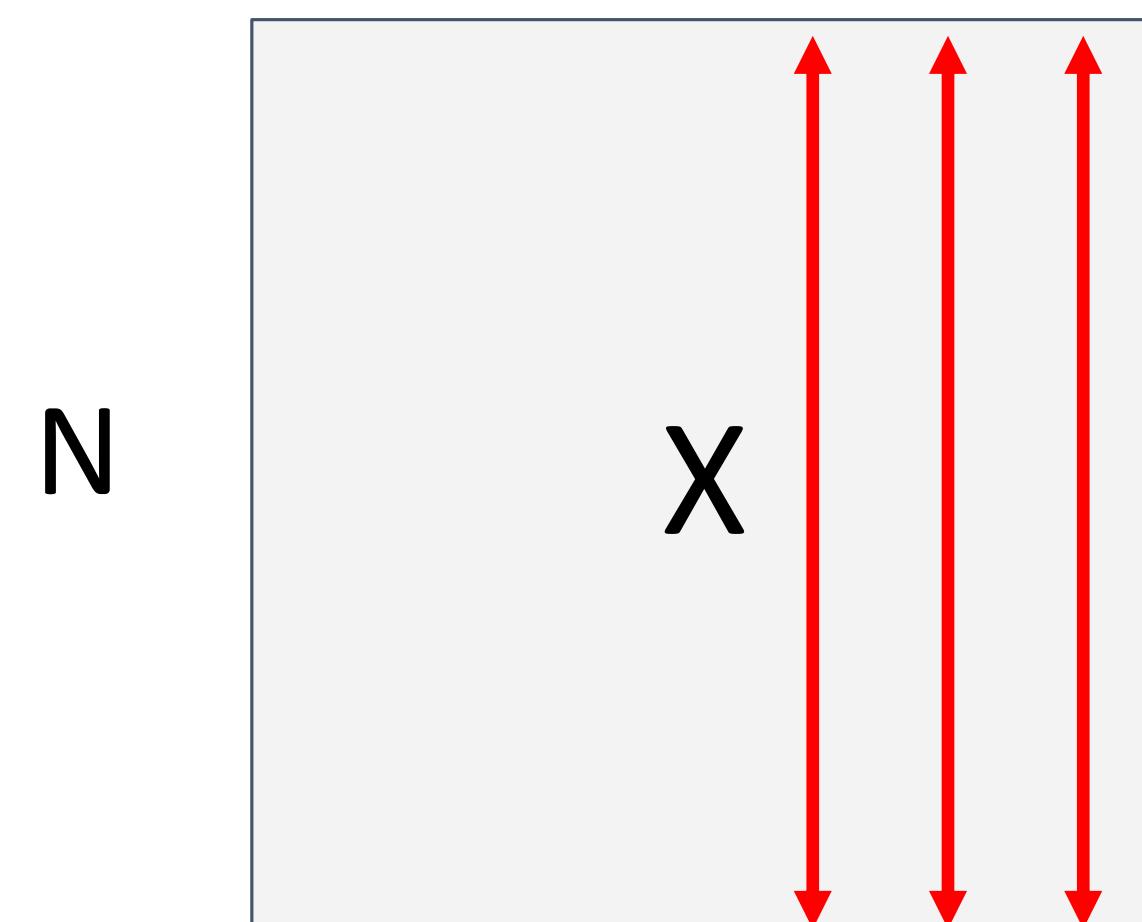
Per-channel
std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is N x D

Batch Normalization

Input: $x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Per-channel
mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Per-channel
std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is $N \times D$

Problem: What if zero-mean, unit variance is too hard of a constraint?

Batch Normalization

Input: $x \in \mathbb{R}^{N \times D}$

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Per-channel
mean, shape is D

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function (in expectation)

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Per-channel
std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,
Shape is N x D



Batch Normalization

Input: $x \in \mathbb{R}^{N \times D}$

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function (in expectation)

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j} \quad \text{Per-channel mean, shape is } D$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2 \quad \text{Per-channel std, shape is } D$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \quad \text{Normalized x, Shape is } N \times D$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j \quad \text{Output, Shape is } N \times D$$

Problem: Estimates depend on minibatch; can't run layer at test-time!



Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of values seen during training

Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

σ_j^2 = (Running) average of values seen during training

Per-channel std, shape is D

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function (in expectation)

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,
Shape is N x D



Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of values seen during training

Per-channel mean, shape is D

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function (in expectation)

$$\mu_j^{test} = 0$$

For each training iteration:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

$$\mu_j^{test} = 0.99 \mu_j^{test} + 0.01 \mu_j$$

(Similar for σ)



Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of values seen during training

Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

σ_j^2 = (Running) average of values seen during training

Per-channel std, shape is D

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function (in expectation)

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,
Shape is N x D



Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of values seen during training

Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

σ_j^2 = (Running) average of values seen during training

Per-channel std, shape is D

During testing batchnorm becomes a linear operator!

Can be fused with the previous fully-connected or conv layer

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,
Shape is N x D



Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

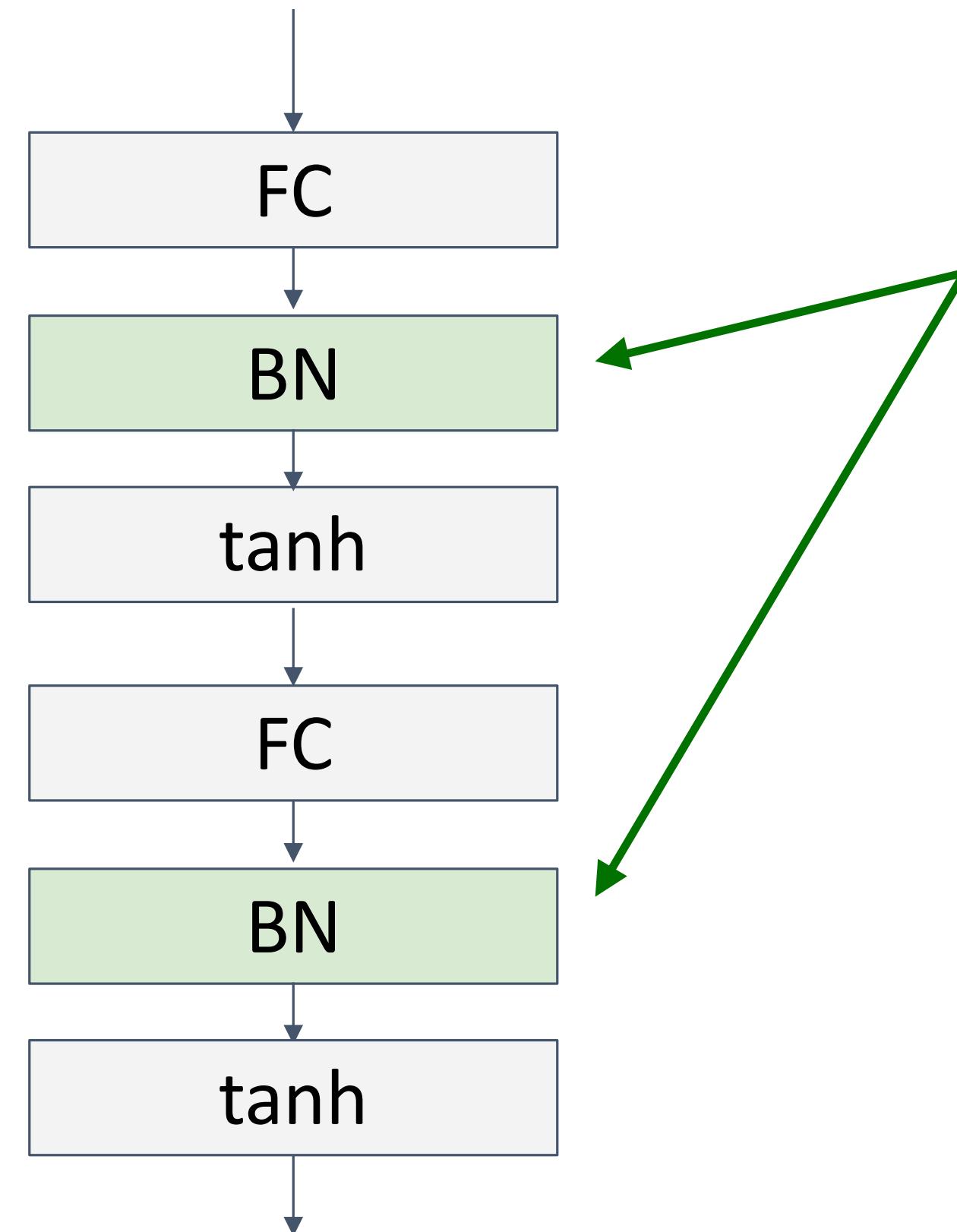
$$\begin{aligned}x &: N \times D \\ \text{Normalize} &\downarrow \\ \mu, \sigma &: 1 \times D \\ \gamma, \beta &: 1 \times D \\ y &= \frac{(x - \mu)}{\sigma} \gamma + \beta\end{aligned}$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$\begin{aligned}x &: N \times C \times H \times W \\ \text{Normalize} &\downarrow \quad \downarrow \quad \downarrow \\ \mu, \sigma &: 1 \times C \times 1 \times 1 \\ \gamma, \beta &: 1 \times C \times 1 \times 1 \\ y &= \frac{(x - \mu)}{\sigma} \gamma + \beta\end{aligned}$$



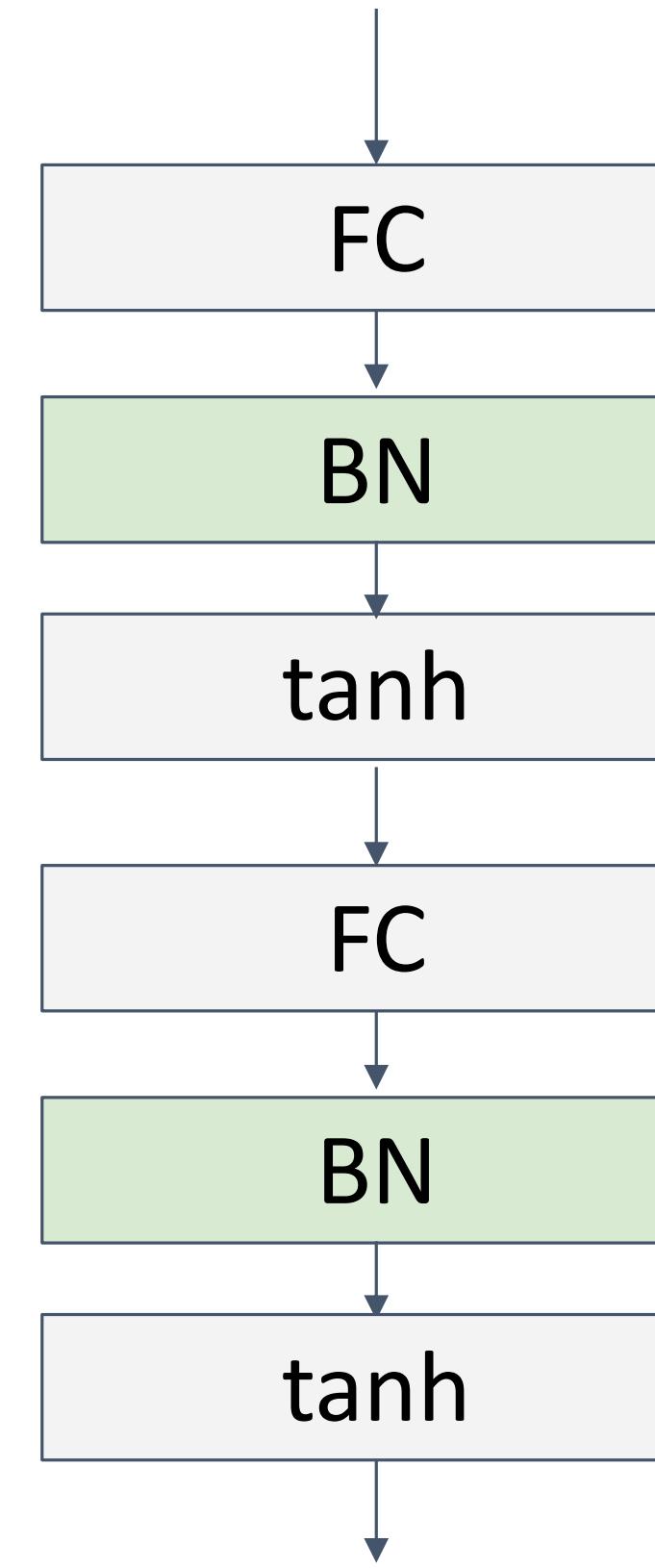
Batch Normalization



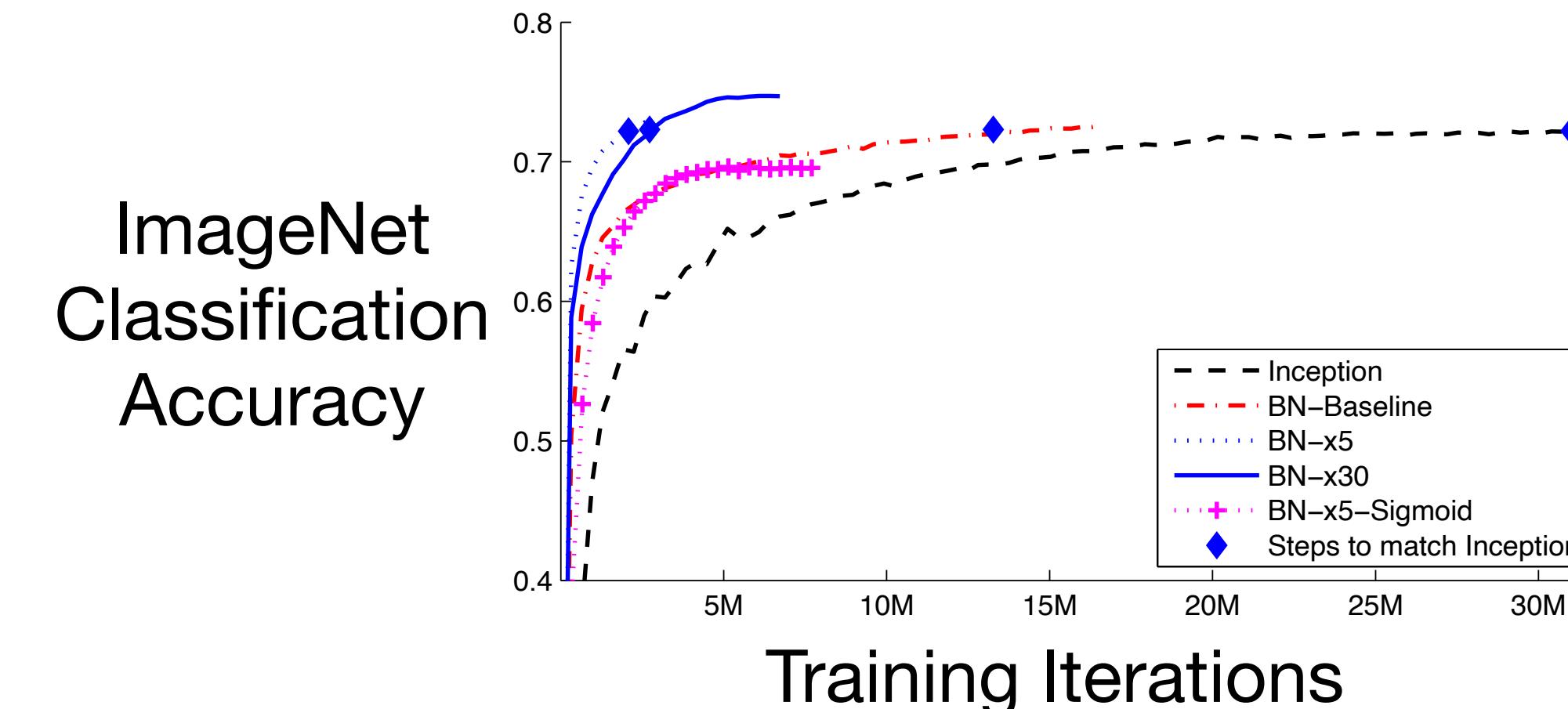
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

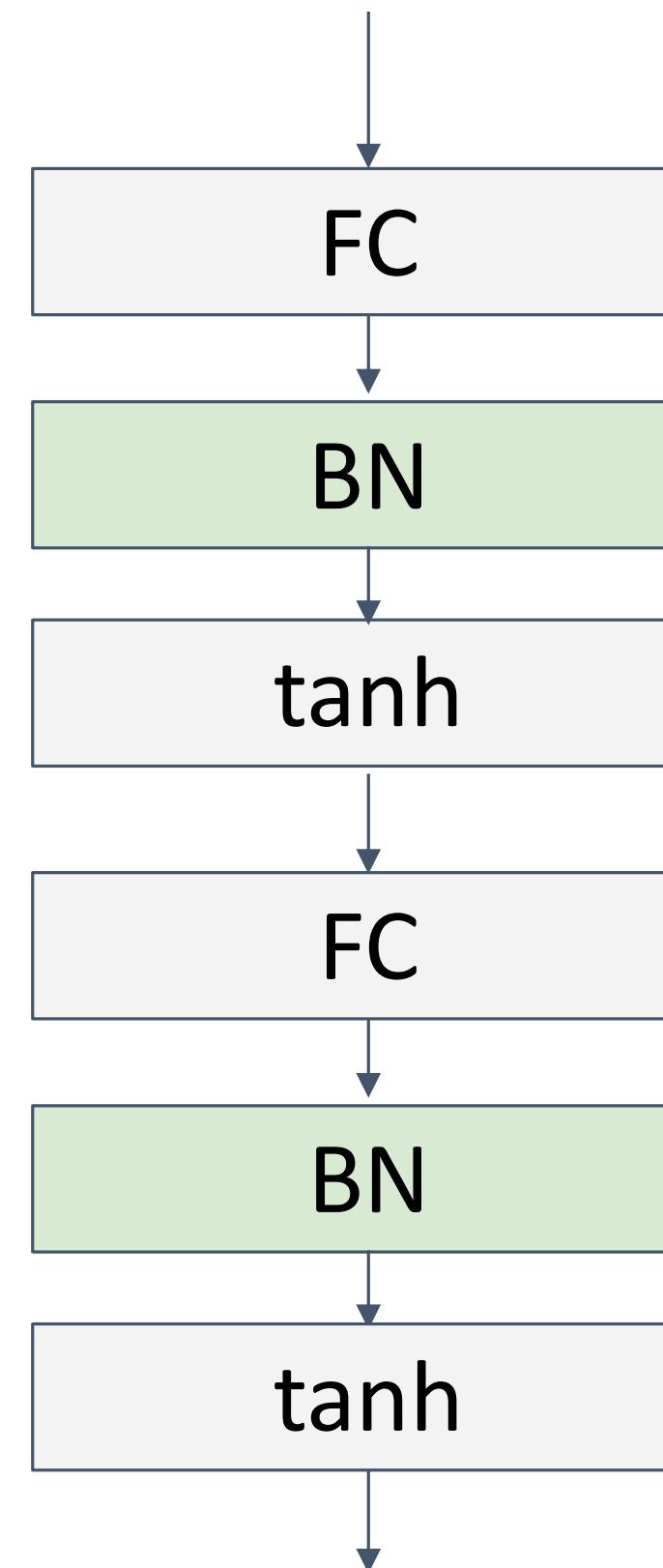
Batch Normalization



- Makes deep networks much easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv



Batch Normalization



- Makes deep networks much easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv
- Not well-understood theoretically (yet)
- Behaves differently during training and testing: very common source of bugs!

Layer Normalization

Batch Normalization for
fully-connected networks

Layer Normalization for fully-connected networks
Same behavior at train and test!
Used in RNNs, Transformers

$$\begin{aligned}x &: N \times D \\ \text{Normalize} &\quad \downarrow \\ \mu, \sigma &: 1 \times D \\ \gamma, \beta &: 1 \times D \\ y &= \frac{(x - \mu)}{\sigma} \gamma + \beta\end{aligned}$$

$$\begin{aligned}x &: N \times D \\ \text{Normalize} &\quad \downarrow \\ \mu, \sigma &: N \times 1 \\ \gamma, \beta &: 1 \times D \\ y &= \frac{(x - \mu)}{\sigma} \gamma + \beta\end{aligned}$$

Instance Normalization

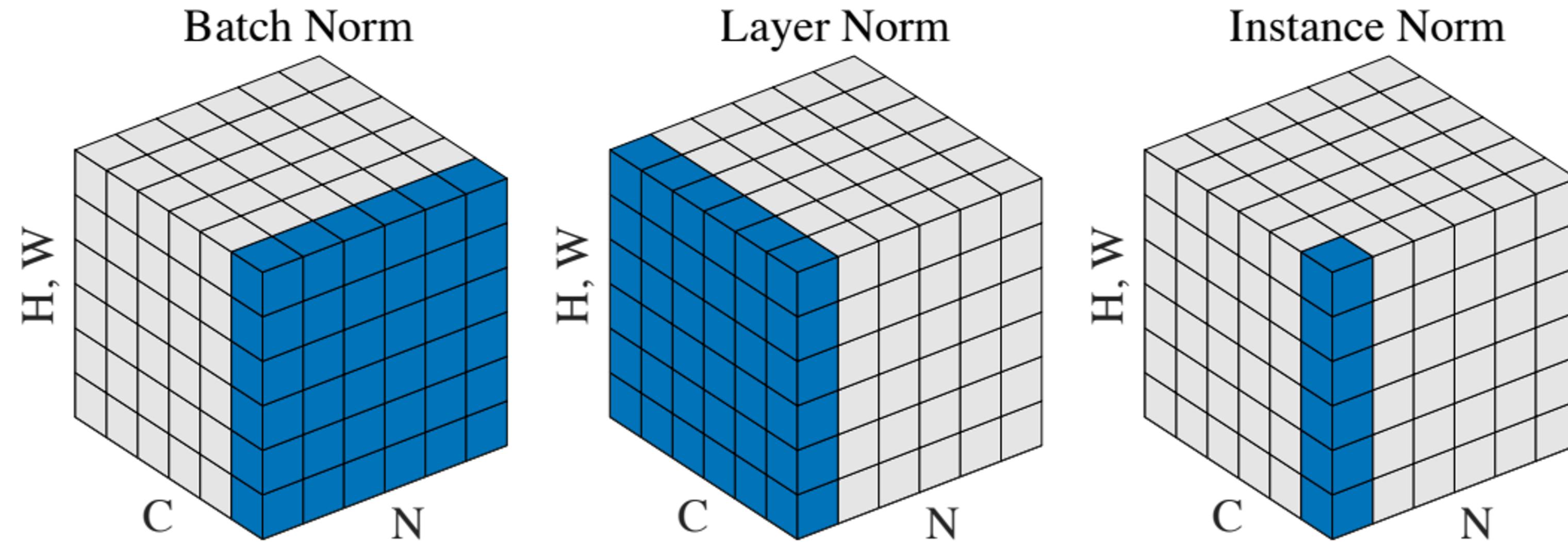
Batch Normalization for convolutional networks

$$\begin{aligned}x &: N \times C \times H \times W \\ \text{Normalize} &\quad \downarrow \quad \downarrow \quad \downarrow \\ \mu, \sigma &: 1 \times C \times 1 \times 1 \\ \gamma, \beta &: 1 \times C \times 1 \times 1 \\ y &= \frac{(x - \mu)}{\sigma} \gamma + \beta\end{aligned}$$

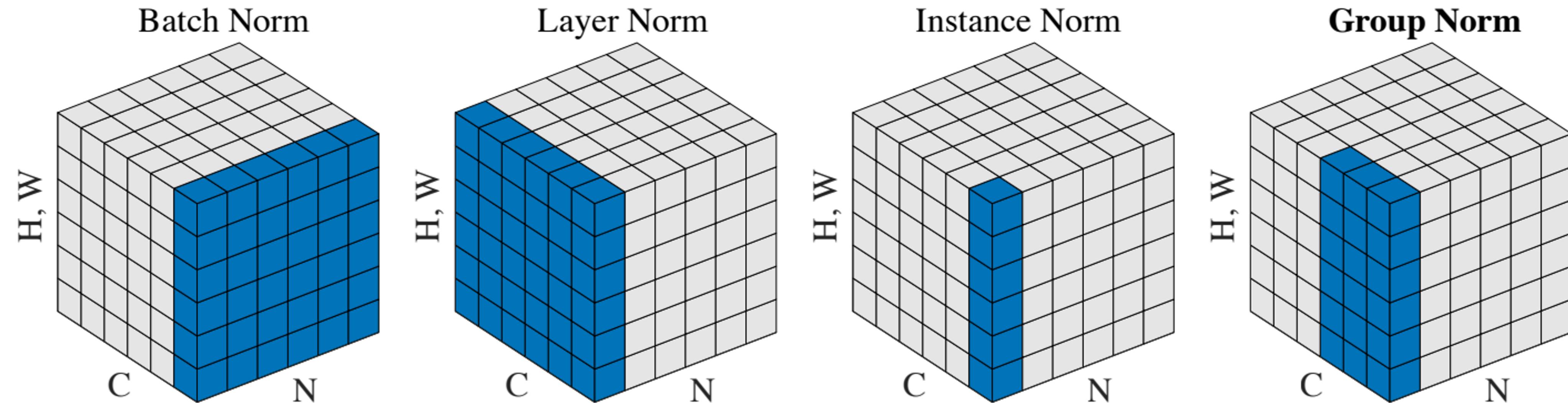
Instance Normalization for convolutional networks

$$\begin{aligned}x &: N \times C \times H \times W \\ \text{Normalize} &\quad \downarrow \quad \downarrow \quad \downarrow \\ \mu, \sigma &: N \times C \times 1 \times 1 \\ \gamma, \beta &: 1 \times C \times 1 \times 1 \\ y &= \frac{(x - \mu)}{\sigma} \gamma + \beta\end{aligned}$$

Comparison of Normalization Layers

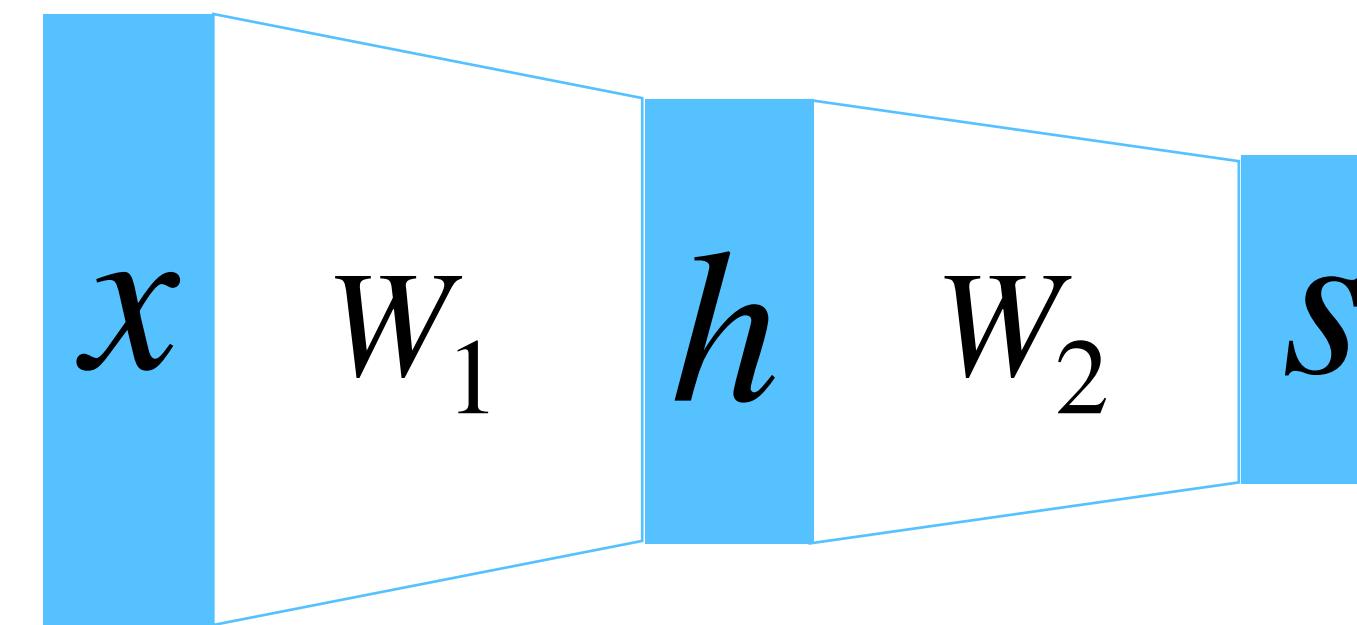


Group Normalization

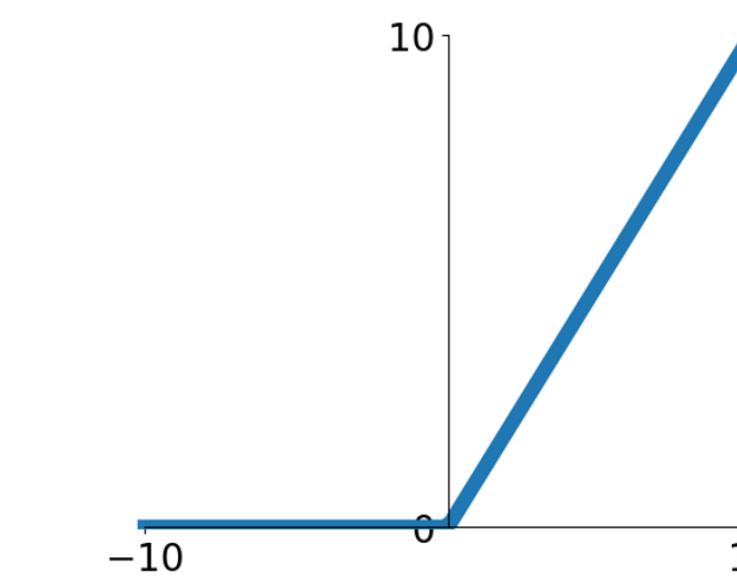


Summary: Components of Convolutional Network

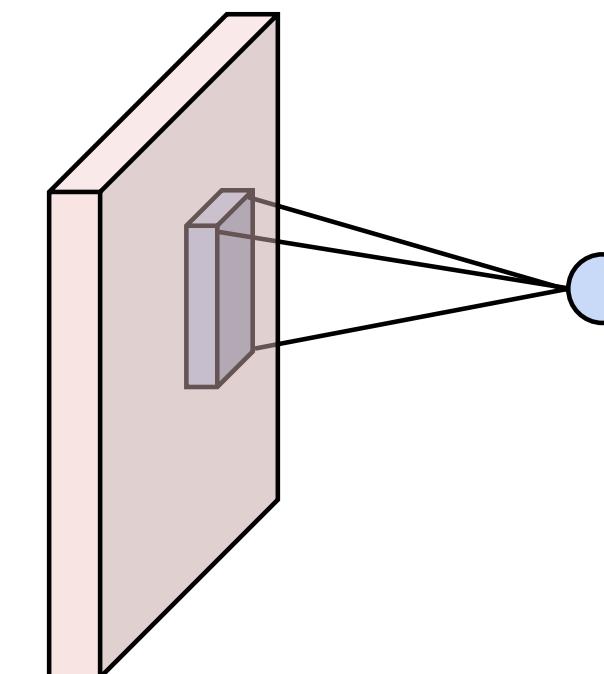
Fully-Connected Layers



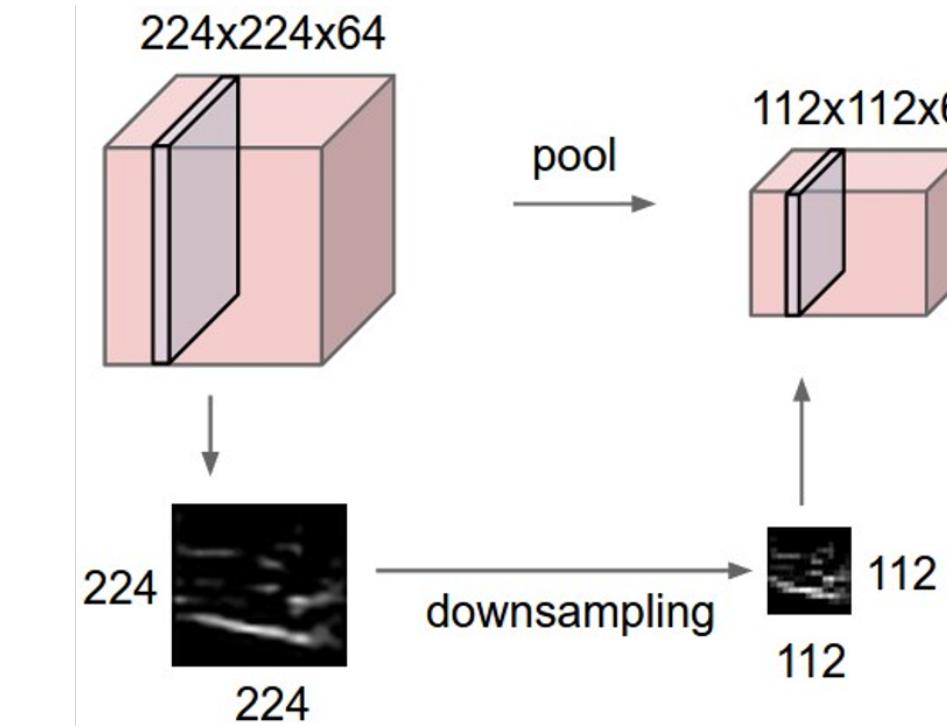
Activation Functions



Convolution Layers



Pooling Layers

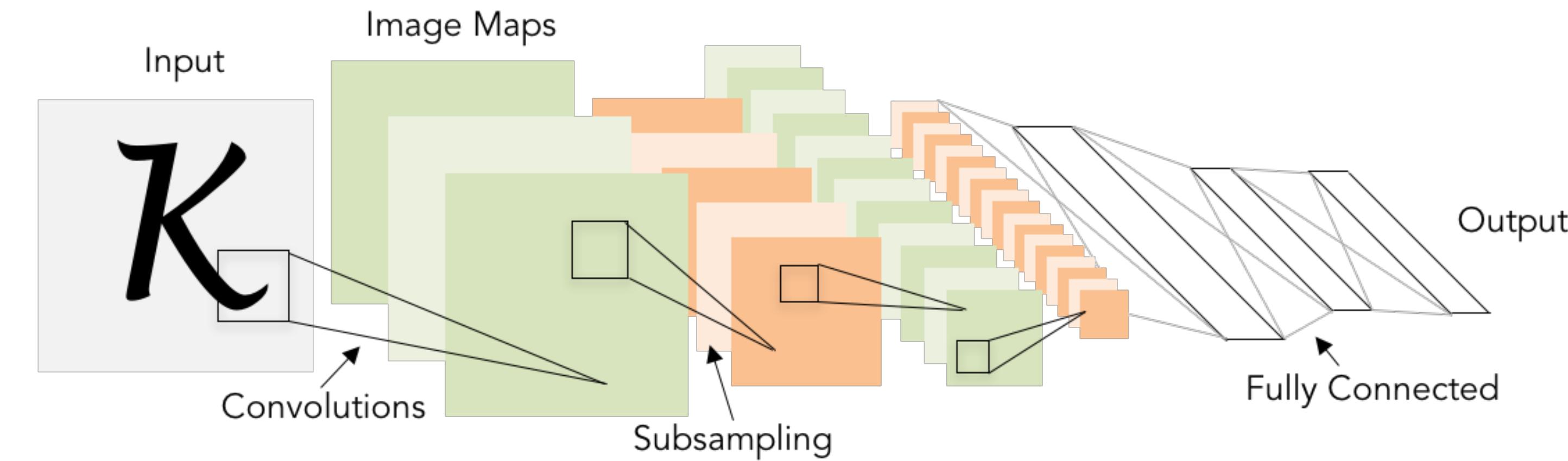


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Summary: Components of Convolutional Network

Problem: What is the right way to combine all these components?



Next time: CNN Architectures



Final Project Overview

- Research-oriented final project
 - Instead of a final exam!
- Objectives
 - Gain experience reading literature
 - Reproduce published results
 - Propose a new idea and test the results!



Final Project Overview

- Research-oriented final project
 - Instead of a final exam!
- Objectives
 - Gain experience reading literature
 - Reproduce published results
 - Propose a new idea and test the results!

Can be completed in teams of 1-3 people

Final Project Deliverables

1. A written paper review
2. In-class paper presentation
3. Reproduce published results
4. Extend results with new idea, technique or dataset
5. Document results in written report



(1) Paper Review and (2) Presentation

Final project teams will be based on overlapping interest

Students will choose from the ‘core’ list of papers on [course website](#)

Each team will be assigned one of the ‘core’ papers to review and present in-class

The 1-page paper review will be due **1-week before** the scheduled presentation

Presentation schedule will be based on paper topic as shown in [course calendar](#)

The screenshot shows a web browser window for 'Papers | DeepRob' at deeprob.org/papers/. The page title is 'Deep Learning Research Papers for Robot Perception'. On the left, there's a sidebar with links: Home, Syllabus, Calendar, Projects, PROPS Dataset, and **Papers** (which is highlighted). Below the sidebar, a note says 'This site uses Just the Docs, a documentation theme for Jekyll.' The main content area has a heading 'TABLE OF CONTENTS' followed by a numbered list of research areas:

- 1 RGB-D Architectures
 - a Core List
 - b Extended List
- 2 Pointcloud Processing
 - a Core List
 - b Extended List
- 3 Object Pose, Geometry, SDF, Implicit surfaces
 - a Core List
 - b Extended List
- 4 Dense object descriptors, Category-level representations
 - a Core List
 - b Extended List

More details on review and presentation criteria in following lectures



(3) Paper Reproduction and (4) Extension

Each team will choose a paper relating to deep learning and robot perception

Doesn't have to be same paper you presented in class

Then reimplement and reproduce at least one of the paper's published results (**not necessarily all the results**)

Then, each team will test one of their own ideas!

By extending the paper's model using new architecture or technique or dataset

Your chance to experiment with deep learning and contribute to the field!

More details on reproduction and extension
in following lectures



(5) Project Report

- The final deliverable for your final project
- A 1-2 page paper
 - What problem within robot perception or manipulation?
 - What work has been done in this area?
 - What approach did you investigate?
 - What questions and directions exist for future work?



More details on report in following lectures

Final Project Grading Overview

- Final Project:
 - Paper Review: 3%
 - Paper Presentation: 3%
 - Paper Reproduction: 6%
 - Algorithmic Extension: 6%
 - Written Report: 6%

More details on report in following lectures



DR



DeepRob

Lecture 7
Convolutional Neural Networks
University of Michigan and University of Minnesota

